E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Regresión Generalizada a través de Programación Genética

Grado en Ingeniería Informática

Trabajo Fin de Grado

Daniel Domínguez Catena

Francisco Javier Fernández Fernández

Pamplona, 26 de Junio de 2014

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Índice general

Resumen

Se propone el diseño, implementación y evaluación de un algoritmo que, a través de técnicas de programación genética y estrategias evolutivas, sea capaz de obtener para una secuencia de números dada una función que la describa de forma ajustada, sin partir de información previa fuera de los elementos de la serie.

El objetivo general es analizar y aproximar funciones complejas y secuencias matemáticas notables, al estilo de la regresión clásica, pero aprovechando la flexibilidad de este tipo de algoritmos para evaluar simultáneamente funciones con formas dispares.

Palabras clave: algoritmo genético, programación genética, estrategia evolutiva, regresión simbólica, regresión

Capítulo 1

Introducción y fundamentos teóricos

1.1. Presentación del problema

Clásicamente se han tratado los problemas de regresión en sus múltiples variantes a través de métodos matemáticos de optimización de parámetros bien conocidos. De esta forma, se plantea una hipótesis (lineal, logarítmica, etc) dependiente de unos parámetros, y se derivan los valores óptimos de estos parámetros para ajustarlos a los datos de entrada[14][6].

Este es un acercamiento al problema que permite derivar de forma muy eficiente y ajustada los parámetros de la solución, al disponerse de una amplia base teórica para estos algoritmos. Además, la naturaleza determinista de la mayoría de estos métodos posibilita acotar de forma certera sus costes y comportamiento, haciéndolos predecibles y regulares en sus resultados.

Sin embargo, en la vida real muchas veces nos planteamos la necesidad de trabajar con expresiones más complejas de las abordables mediante regresión clásica. Es habitual encontrarse con datos que no se corresponden con un patrón conocido y bien estudiado en regresión, o que directamente no se ajustan de forma clara a ningún tipo de estructura. De esta forma, perdemos la oportunidad de estudiar este tipo de datos con las herramientas comunes y automatizadas, viéndonos forzados a la observación directa como método de estudio.

En general, podemos pensar en problemas de la vida real tan dispares como el análisis de resultados bursátiles, la predicción meteorológica o el amplio campo de las sucesiones matemáticas.

En este tipo de situaciones partimos de información incompleta, por lo que los algoritmos de optimización clásicos, que habitualmente requieren de fuertes condiciones de regularidad en los datos, no proporcionan una buena solución. Sin embargo, dado que los algoritmos genéticos son herramientas idóneas para trabajar con información desestructurada e incompleta, en esta memoria consideramos su utilización para tratar este tipo de problemas.

Ahora bien, puesto que el problema en toda su complejidad puede resultar inabordable, trataremos una versión más específica. En concreto, decidimos restringirnos al caso de las secuencias matemáticas, de forma que nos planteamos la posibilidad de implementar un algoritmo genérico capaz de, dada una cierta secuencia de números reales, obtener una fórmula que la aproxime, sin datos previos sobre la secuencia en cuestión más allá de los valores de sus elementos.

La justificación de esta elección viene dada por la naturaleza de estas secuencias. En general, al venir de cuerpos matemáticos conocidos, dispondremos en muchos casos de la fórmula original para poder comprobar nuestras soluciones. Además, trataremos con datos sin ausencias, bien ordenados, y en ausencia de ruido. En general, esto nos permitirá disponer de una gran cantidad de casos de estudio muy dispares entre si, lo que nos facilitará localizar los tipos de secuencias para los que el algoritmo propuesto sea potencialmente más útil.

1.2. Especificación del problema

Partiremos de secuencias de números reales, con el objetivo de determinar una expresión capaz de replicar la secuencia original por completo (y previsiblemente los siguientes elementos de ésta).

Las pruebas del algoritmo se centrarán en el segmento de las sucesiones matemáticas clásicas, normalmente compuestas por enteros, al ser en este campo donde más útil puede resultar este algoritmo.

En general, no buscamos "competir" con los algoritmos específicos de regresión de cada modelo (regresión lineal, polinomial, logarítmica, etc), siendo el objetivo principal no la velocidad del algoritmo si no su capacidad para resolver funciones genéricas.

Nos centraremos por tanto en sucesiones matemáticas clásicas, construidas de forma algorítmica (computables por un ordenador) y libres de error. Además, supondremos que son completas, esto es, que dispongamos de todos los elementos de la serie perfectamente ordenados, sin elementos ausentes o fuera de lugar.

Buscaremos como solución una fórmula simple que aproxime al máximo la secuencia dada. En particular nos interesarán las respuestas que ajusten la secuencia al completo con un error mínimo, pero consideraremos que cualquier buena aproximación revela un cierto patrón o tendencia de la secuencia original, y será por tanto un resultado razonable. Esto será particularmente importante para secuencias "intratables", como las derivadas de los números primos, en las que localizar cualquier subserie o regularidad resulta ya de gran interés.

Las fórmulas resultantes estarán formadas sobre un lenguaje de operadores aritméticos habituales, comprendiendo también símbolos para constantes, una variable tiempo y variables que permitan referenciar a los valores anteriores de la serie.

La inclusión de variables que hagan referencia a valores previos de la serie se corresponde con las funciones recursivas, muy habituales en el tipo de problema que vamos a abordar, y muchas veces imposibles de reconvertir a una forma explícita no recursiva (valga como ejemplo la conjetura de Collatz, donde está

aún por demostrarse si la ecuación recursiva es igual a la forma explícita trivial 1[13]).

1.3. Campo de aplicación

Como ya se ha mencionado, el algoritmo se centrará en la regresión de sucesiones matemáticas clásicas. Si bien este es un campo típicamente considerado más cercano a la teoría matemática (teoría de números) que a la aplicada, en los últimos años ha adquirido una gran importancia. Por una parte, el análisis de series temporales ha adquirido una gran relevancia en los últimos años, y por otra parte es una de las bases sobre las que se fundamenta la criptografía, que a su vez constituye uno de los grandes motores de las matemáticas modernas.

En concreto, la impredecibilidad de la secuencia de los números primos, así como de múltiples variantes y derivadas, constituye el principal cimiento de los métodos criptográficos empleados en la actualidad. Junto con los métodos basados en curvas elípticas, estas secuencias son las principales generadoras de números pseudoaleatorios empleadas en ordenadores.

En base a esto, consideramos de gran interés el disponer de métodos capaces de analizar estas secuencias y arrojar patrones y resultados sobre ellas, facilitando nuestra comprensión de su funcionamiento interno.

En el caso del algoritmo propuesto, nos centraremos en problemas más simples (donde existe algún patrón ya conocido), principalmente por limitaciones hardware. Aun así, los resultados aquí expuestos podrían extrapolarse a este tipo de secuencias más complejas, de disponer del equipo necesario para aplicarlas.

1.4. Métodos clásicos

Los métodos clásicos de tratamiento de los problemas de regresión más complejos son principalmente la regresión no lineal y la regresión segmentada.

- La regresión no lineal se basa en el ajuste de parámetros de funciones no lineales, permitiendo una expresividad similar a la que nos proponemos alcanzar en esta memoria. Sin embargo, sigue siendo necesaria la intervención humana para determinar el modelo de función a emplear.

  Existen varios libros dedicados al tema, entre los que destaca la excelente introducción de (Jianqing 96[7]).

- Por otra parte, la regresión segmentada se basa en la descomposición del problema en el espacio de las variables .explicativas"(en nuestro caso de estudio, la posición del elemento en la secuencia). Este método permite considerar distintos comportamientos de los datos a trabajar en distintos fragmentos de la secuencia inicial, permitiendo así emplear métodos simplificados de regresión en cada fragmento.

  Se puede encontrar una explicación más detallada de la técnica, junto con un ejemplo aplicado y una comparativa con otros métodos en (Ritzema 94 [16]).

Si bien existen múltiples métodos de análisis de regresión, ampliamente estudiados en el campo de la estadística, estos dos son los que más nos interesan para nuestro trabajo. El lenguaje que implementaremos para modelar nuestras soluciones, que será presentado más adelante en esta memoria, busca una expresividad similar a la de la regresión no lineal, pero pudiendo aprovechar las posibilidades de descomposición propias de la regresión segmentada.

1.5. Estado del arte

El problema propuesto recibe actualmente el nombre de *regresión simbólica* en la mayoría de las fuentes. En general, se trata de un campo de reciente aparición. En concreto, las bases se remontan a finales de los años 80 aproximadamente, donde (Diday 1987a [2] y 1987b[3]) define los principios de esta aproximación al problema de la regresión generalizada o simbólica.

Actualmente se han desarrollado varias herramientas dedicadas a la regresión simbólica (de entre las cuales podemos destacar Eureqa [5] y Sodas [20] principalmente) y han sido aplicadas con cierto éxito ([11], [22] y [4] por ejemplo).

Los últimos avances en la materia hacen referencia al estudio de complejidades en los resultados, una de las principales taras de este método de descubrimiento de funciones. En concreto, se puede mencionar el trabajo de [23], donde se estudia la implementación del orden de no linearidad como medida de complejidad de modelos generados por regresión simbólica.

1.6. Novedades de la propuesta

Como hemos visto en la anterior sección, los métodos de regresión simbólica ya han sido implementados en varias ocasiones en múltiples variantes. En concreto, en nuestra implementación nos centraremos en dos posibles mejoras a los métodos ya existentes: la combinación de programación genética con estrategias evolutivas (de forma que subdividamos el problema en dos partes, la derivación de la fórmula y la optimización de constantes) y la inclusión de variables temporales (referentes a los valores anteriores de la secuencia, de forma que generalicemos el análisis de series temporales).

1.6.1. Combinación de programación genética y estrategia evolutiva

En general, podemos decir que uno de los puntos vitales de toda fórmula son sus constantes, hasta el punto de que la mayor parte de los métodos de regresión clásicos ajustan únicamente estas constantes, y no los operadores o relaciones de la fórmula.

Dentro de la regresión simbólica ha sido ya estudiada la posibilidad de mejorar los algoritmos comunes, basados en programación genética exclusivamente, hibridándolos con métodos de optimización de parámetros. La idea general suele ser

permitir al algoritmo genético derivar una fórmula plausible (a nivel de símbolos y relaciones), y después mejorarla por optimización de parámetros, de forma sistemática, nada más generarse la fórmula (antes de que vuelva al ciclo común de la programación genética y se encuentre con el resto de individuos de la población)[9].

Esta implementación permite al algoritmo de programación genética que deriva la fórmula ignorar por completo la existencia de parámetros, especializándose en su tarea independientemente de la labor de optimización del otro algoritmo.

En concreto, para optimización se han estudiado los siguientes algoritmos:

- El algoritmo STROGANOFF de (Iba, Sato y de Garis, 95[9]) emplea el método GMDH ("Group method of data handling" propuesto por (Ivakhnenko 70[10]).
- (McKay et al. 95[15]) emplea optimización por mínimos cuadrados no lineales sobre cada constante nueva que aparece en el cuerpo de fórmulas. En este caso trabaja con un operador de mutación particular que transforma las variables normales en constantes y viceversa, de forma que se tiene "localizada" la aparición de nuevas constantes y pueden optimizarse ya en su contexto final.
- (Schoenauer, Lamy, y Jouve 95 [17]) emplearon un método típico de optimización por descenso por gradiente, integrado en un operador de mutación en este caso.
- (Sharman, Esparcia Alcazar, y Li 95 [19]) emplearon un método de "simulated annealing" ("recocido simulado").

Nuestra propuesta será, en vez de emplear estos métodos clásicos, utilizar una estrategia evolutiva para realizar esta optimización. Siendo la ventaja principal de todo algoritmo genético su robustez y la capacidad de explorar grandes espacios de soluciones (Sivanandam 07[18]), el uso de un método de estrategia evolutiva nos permitirá cubrir un gran espacio de soluciones. Al estar trabajando con secuencias de números de orígenes diversos, y con funciones de formas variadas, el poder optimizar cualquier función, aunque no sea a sus valores óptimos, nos permitirá recorrer este espacio de soluciones con mayor eficacia.

El mayor problema a abordar aquí será que, dada la naturaleza impredecible de las estrategias evolutivas, no dispondremos de un indicador claro de cuándo hemos obtenido un valor óptimo en nuestra solución. La forma de reducir este efecto será ejecutar periódicamente la optimización de parámetros, de forma que las soluciones que más tiempo sobrevivan en la programación genética realicen más ciclos de optimización. De esta forma, la solución final vendrá de un largo proceso de optimización, sin que perdamos un tiempo excesivo en la optimización de soluciones parciales o individuos no aptos.

1.6.2. Inclusión de variables temporales

En general, los algoritmos de regresión suelen explicar una cierta variable en función de otras, construyendo una fórmula que las asocie ($y = f(x, z, w, \ldots)$).

Otro de los casos habituales es el análisis de series temporales, donde se relaciona la variable objetivo con sus anteriores valores en la serie de estudio ($x_t = f(x_{t-1}, xt-2, \ldots)$).

En nuestra implementación, incluiremos variables de los dos tipos. Específicamente, como vamos a tratar sucesiones simples de números, para la regresión general tendremos sólo una variable objetivo y una única variable temporal, t, sobre la que explicarla. Del análisis de series temporales tomaremos las variables referentes a los anteriores valores de la serie, limitando la cantidad de elementos que podemos retroceder (para reducir el espacio de soluciones y evitar que haya demasiados elementos no evaluables por no disponer de suficientes elementos detrás).

Es importante destacar que nos separaremos del análisis de series temporales (así como de otras implementaciones de regresión simbólica, como Eureqa[5], que ya han manejado variables temporales) en un detalle crucial. Donde habitualmente se toman para cada elemento como previos los de las posiciones anteriores de la sucesión original, nosotros tomaremos los elementos previamente predichos.

En la práctica, esto significa que podremos emplear una cierta "semilla" recursiva (si permitimos que las fórmulas miren 4 elementos hacia atrás, por ejemplo, será un vector con los 4 primeros elementos de la serie), análoga a los casos triviales de cualquier fórmula recursiva, y a partir de ahí construir la totalidad de la secuencia con la fórmula resultante.

Pensemos en una cierta función por tramos, por ejemplo, que evalúe a una constante c_0 en todos los elementos de la serie antes de un cierto umbral, y a una segunda constante c_1 de ahí en adelante. El análisis de series temporales clásico decidirá con seguridad que, para cada elemento, este es igual al anterior, acertando con esta predicción todos los elementos salvo el inmediatamente posterior al umbral. Si quisiésemos construir la función recursiva correspondiente, sin embargo, se perdería la información del segundo tramo y evaluaría siempre a c_0.

En nuestra implementación, sin embargo, calcularemos el error sobre la función recursiva resultante, por lo que exigiríamos a las soluciones que incluyesen la información de ambos tramos.

Capítulo 2

Diseño

2.1. Estructura general del algoritmo

Primero, es interesante considerar el funcionamiento de un algoritmo genético genérico, que será la base del nuestro.

En la Figura 2.1 hemos considerado una ligera variante del algoritmo genético clásico, donde normalmente se comprobaría desde la primera población inicial si se cumple o no con el criterio de salida. Por asemejarlo con nuestro algoritmo, donde esto no es así (por cuestiones de implementación, simplemente), hemos considerado esta pequeña variación.

En general, un algoritmo genético es una herramienta simple donde depuramos una cierta población de individuos, mezclándolos y modificándolos de forma más o menos arbitraria y empleando una serie de herramientas para lograr que las sucesivas generaciones converjan hacia un resultado óptimo.

En nuestra versión, el funcionamiento será el mismo, pero añadiremos un nuevo paso, como se puede ver en la Figura 2.2.

El paso indicado comprenderá la ejecución de una estrategia evolutiva reducida que optimice los parámetros de cada uno de los individuos de la población. En general, todo individuo realizará en cada iteración del algoritmo principal una optimización, aunque existirán una serie de parámetros de control para omitir algunas pasadas y reducir la carga computacional del algoritmo.

2.2. Estructuración y modularización del diseño

Si bien la idea del algoritmo es relativamente simple, la implementación es considerablemente laboriosa y compleja. Para facilitar la comprensión del código y facilitar la comprobación de que cada parte individual funciona correctamente, emplearemos la siguiente estructura basada en módulos:

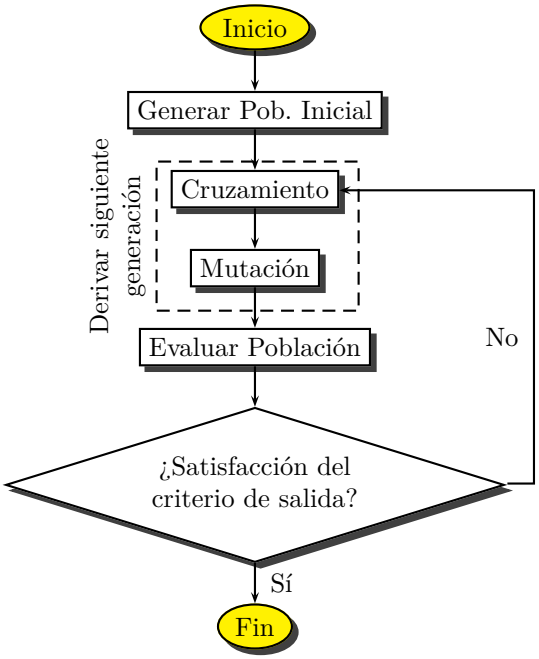

Figura 2.1: Algoritmo genético genérico, Diagrama de flujo general

- **Main**, la función principal que gestionará la interfaz de ficheros (a través de los cuales le indicaremos al programa la labor a realizar) y lanzará la regresión.
- **Parámetros**, que gestionará un diccionario de parámetros del que hará uso el resto del programa, permitiendo personalizar cada ejecución del algoritmo a los datos que queremos estudiar.
- **Dirección**, que ejecutará el cuerpo de la regresión.
- **Evaluación**, que, dado una cierta función, calculará el valor de la función de *fitness* (error) de esa función para la secuencia objetivo.
- **Pila de cálculo**, que implementará una pila de cálculo para notación postfija sobre nuestro lenguaje aritmético propuesto (este lenguaje se detallará en la Sección 3.3).
- **Generación**, que contendrá los métodos de producción de fórmulas aleatorias.
- **Torneo**, que implementará el método del torneo para la selección de progenitores.
- **Programación Genética**, que contendrá los métodos específicos de la programación genética: mutación, cruzamiento y generación de la población inicial.

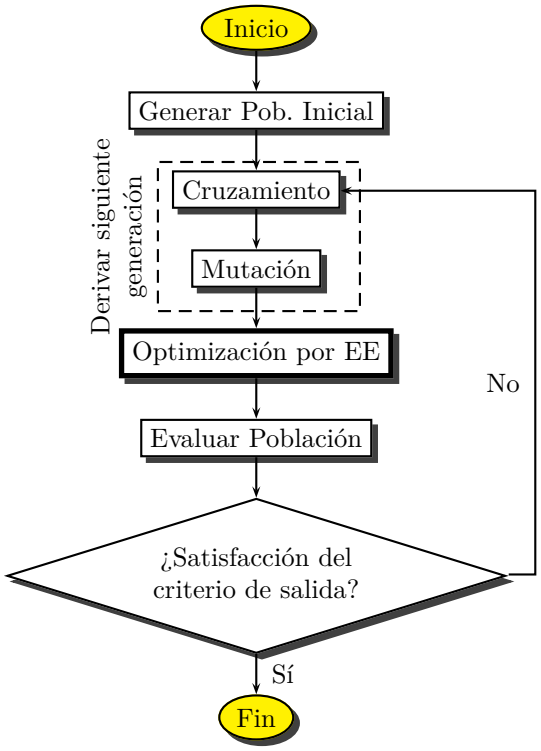

Figura 2.2: Algoritmo propuesto, Diagrama de flujo general

- **Estrategia Evolutiva**, que contendrá los métodos específicos de la estrategia evolutiva: mutación y generación de la población inicial.

Además, manejaremos algunos módulos con detalles de implementación, irrelevantes a la estructura real del algoritmo.

- **Constantes**, que contendrá las constantes correspondientes a los símbolos de nuestro lenguaje.
- **Tipos**, que contendrá la descripción de los tipos de las diferentes poblaciones y fórmulas e individuos (tendrán tipos diferentes en las distintas fases del algoritmo).
- **Gráficos**, que implementará las funciones para gestionar la salida de datos en gráficas, tanto para mostrar por pantalla como para escribir a ficheros.
- **Funciones auxiliares**, que contendrá el resto de pequeños fragmentos de código que vamos a emplear en la implementación.

Atendiendo a esta subdivisión más específica, el diagrama de flujo elaborado será similar al de la Figura 2.3.

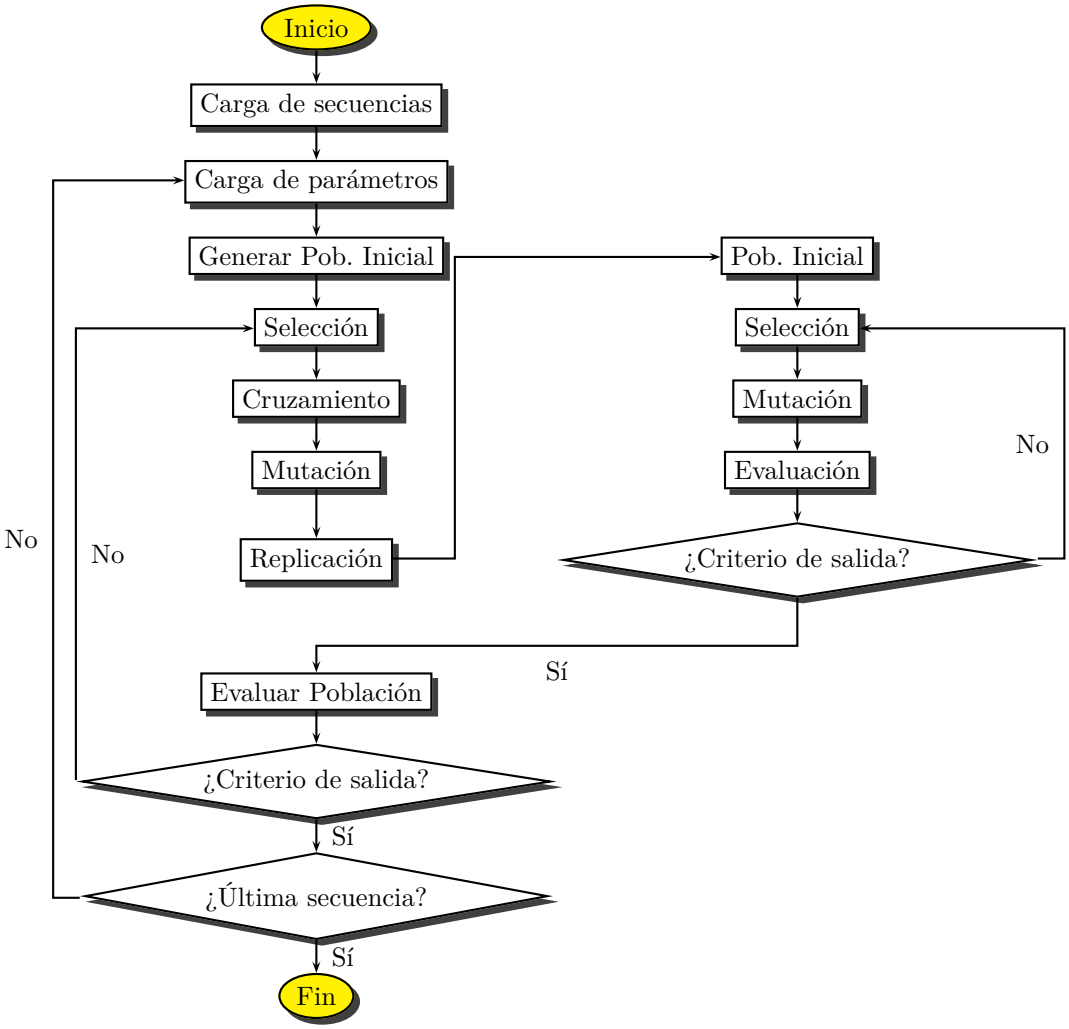

Figura 2.3: Algoritmo propuesto, Diagrama de flujo específico

Capítulo 3

Implementación

3.1. Elección del lenguaje de programación

Elegimos como lenguaje para la implementación del algoritmo C++.

Para esta elección, hemos tenido en cuenta varias cuestiones:

- Eficiencia del lenguaje. Para un algoritmo computacionalmente intensivo conviene buscar un lenguaje compilado puro (no interpretado ni semicompilado, como pueden ser Java o Python, por ejemplo), y con gestión manual de memoria. La cuestión de la memoria es particularmente importante, ya que vamos a trabajar con cientos de arrays de pequeñas dimensiones que tendremos que crear y destruir prácticamente en cada iteración, por lo que retrasar un pequeño tiempo la liberación de memoria (como suelen hacer los recolectores de basura automáticos) implicaría aumentar considerablemente el consumo de memoria del programa.

    - C++ es conocido por su eficiencia, heredada de C, y dispone de todas las herramientas típicas de su hermano mayor.

- Estructuras preimplementadas. Aunque para los puntos críticos del sistema emplearemos arrays de bajo nivel (tipo C) para evitar sobrecostes, para estructuras grandes (como las poblaciones, por ejemplo) resulta muy conveniente disponer de estructuras ya implementadas como listas, árboles, mapas, etc. Además, esto nos ofrecerá la posibilidad de utilizar herramientas extra como algoritmos de ordenación, copia, etc. ya conocidos y bien optimizados.

    - C++ dispone de una librería estándar amplia y bien construida, con implementaciones muy eficientes de miles de algoritmos y estructuras clásicos.

- Existencia de técnicas de paralelismo, multiescalaridad, clusterización... Al tratarse de un algoritmo genético en general, y en particular debido a nuestra estructura de programación genética + estrategia evolutiva,

algo como el paralelismo a nivel de procesador mejoraría enormemente la velocidad del algoritmo.

- C++ dispone de todas las herramientas que podamos necesitar en este sentido, desde CudaC (paralelismo sobre GPU) hasta OpenMP (paralelismo a nivel de procesador), pasando por acceso a las capacidades SIMD de los procesadores Intel (paralelismo a nivel de instrucción).
- En nuestro caso, implementaremos una solución escalar paralela a nivel de procesador mediante OpenMP, dada la relativa simplicidad de la librería y su perfecta adaptación al problema.

3.2. Representación de los datos

Para la representación de los datos a examinar optamos por un array clásico estilo C. Específicamente, emplearemos un *struct* de C que contendrá tanto la longitud de la entrada como el puntero al primer elemento.

El objetivo de esta elección es maximizar la eficiencia, al ser este juego de datos uno de los arrays más utilizados del código (sobre el que iteraremos repetidas veces).

Los datos serán de tipo *xreal*. Definiremos el tipo en cuestión como *double* o *float*, según la precisión requerida para la entrada en cuestión (según el máximo elemento de la serie, principalmente). En general trabajaremos con *double*, ya que experimentalmente hemos comprobado que no supone un gran problema de eficiencia y mejora la precisión considerablemente en las operaciones internas.

Es importante destacar que no podríamos trabajar a priori sobre enteros, pese a ser los principales objetivos de nuestro trabajo (la mayor parte de las secuencias notables se componen de enteros o racionales que podrían reconvertirse en secuencias de enteros alternando numeradores y denominadores). La razón es que muchas de estas secuencias trabajan con enteros de grandes órdenes de magnitud, muy por encima de lo que el tipo *int* estándar puede representar. Sí que sería razonable plantearse una implementación con enteros grandes, a través de librerías externas, pero consideramos que se sale del objetivo de este trabajo.

3.3. Representación de las soluciones

El lenguaje que emplearemos se compondrá de:

- Operadores
  - Suma y resta, denotadas por `+` y `-`
  - Producto, denotado por `*`
  - Cociente protegido, denotado por `/`

    - La protección se refiere a que retornaremos 0 en los casos no determinados (división por 0 y la indeterminación 0/0). La razón de esto es ofrecer al lenguaje formas fáciles de obtener la constante 0, así como de evitar que se pierdan individuos de alta calidad debido a un único caso de división por 0 en el rango del problema. Puede
  - Potencia, `^`
    - Esta operación también cumplirá las funciones de las raíces, a través de la igualdad $\sqrt[n]{x} = x^{\frac{1}{n}}$.
  - Comparación, denotada por `>`
    - Retornaremos 0 si el resultado es falso, y 1 si es verdadero.
    - Sólo necesitamos implementar la comparación en un sentido, ya que la inversa (menor que, en este caso) se obtiene invirtiendo las ramas hijas.
    - No consideramos de interés la comparación de igualdad (`=`) al trabajar en el espectro de los números reales. El incluirla sólo incrementaría el tamaño del espacio de soluciones, y en general acabaría siendo utilizado como un costoso sustituto del 0.
  - Módulo, denotado por `%`
    - Es interesante incluir la operación módulo, ya que en combinación con el terminal tiempo 'T' nos ofrecerá una forma de aplicar transformaciones en intervalos periódicos de la solución.
    - Dada la naturaleza entera de la operación módulo, optamos por realizar sobre ambos operadores la función suelo antes de aplicar el módulo. En general, hemos decidido que el objetivo de esta función será permitirle al algoritmo resolver periodicidades, por lo que este paso a entero facilitará que la constante que indique la periodicidad no requiera una gran precisión (para indicar periodicidad 3 nos servirá una constante real con cualquier valor entre 3 y 4, por ejemplo).

- Terminales
  - Variable tiempo, `T`.
    - Esta variable retornará la posición del elemento en cálculo dentro de la secuencia. Esto es, sería la variable x usada comúnmente en la representación formal de cualquier secuencia matemática.
  - Variables referentes a los anteriores elementos de la serie, denotadas por `P[n]`, donde `n` es un número natural entero mayor que 1. `n` valdrá 1 para referirse al anterior elemento de la serie del que estamos calculando, 2 para el anterior a éste, y así sucesivamente.
  - Constantes `0` y `1`.
  - Constantes arbitrarias, representadas por `K[n]`, donde `n` es el valor particular de la constante. Estas constantes son las que ajustaremos mediante estrategias evolutivas.

En general se consideran dos cuestiones principales para la implementación de un lenguaje para programación genética, consistencia de tipos y seguridad de evaluación[12][21].

- La consistencia de tipos se refiere a la compatibilidad entre las operaciones. Este es un problema habitual en programación genética clásica, donde el comprobar que una instrucción tenga variables sólo de los tipos adecuados puede ser considerablemente costoso. En nuestro caso, al partir de un lenguaje sólo aritmético, y disponiendo de un tipo universal `xreal`, tenemos esta consistencia garantizada. La única operación "conflictiva", el módulo, realiza sus conversiones internas y se adapta al tipo xreal.
- La seguridad de evaluación se refiere a la posibilidad de realizar operaciones fuera de rango, divisiones por cero y otras operaciones de resultados no determinados.
  - Para las operaciones simples (como el cociente), hemos implementado guardas para los casos comunes (división por cero y similares), de forma que la evaluación siga siendo válida.
  - Para el resto de casos infrecuentes (desbordamientos en potencias, logaritmos negativos y similares), toleramos que el resultado sea erróneo en la operación. Hemos colocado guardas generales en el retorno final de la evaluación completa, de forma que si se da alguno de estos casos desechemos la fórmula de forma segura.

    En general esta solución no suele ser deseada por la posibilidad de eliminar individuos generalmente aptos, pero incluir guardas para cada posible operación aritmética errónea es complejo e incrementa el coste computacional de cada evaluación (al tener que realizar comprobaciones extra sobre el resultado).

Para la representación de las soluciones al problema (fórmulas), existen dos opciones principales:

- Representación en forma de árbol
  - En esta representación cada nodo representará una operación, y en las hojas colocaremos los terminales del lenguaje.
- Representación en forma de vector
  - En esta representación empleamos una notación alternativa (de postfijo o de sufijo, frente a la común de infijo) para poder escribir la fórmula linealmente sin ambigüedad.

La primera es la más natural y directa para una fórmula, pero no está exenta de problemas. En particular, la representación del árbol suele exigir el empleo de punteros a las hojas, lo que reduce la eficiencia del recorrido del árbol.

En nuestro caso la velocidad es crucial, ya que tendremos que evaluar la fórmula al menos una vez para cada elemento (bien de la población de programación

genética o bien de la de estrategia evolutiva), y cada evaluación implicará aproximadamente n recorridos sobre la fórmula (donde n es la longitud del vector de datos de entrada). Debido a esto, consideramos más adecuada la segunda implementación.

La contrapartida a tener en cuenta es que, si no queremos redimensionar los arrays sobre la marcha (conforme las fórmulas crezcan o cambien), nos veremos obligados a reservar desde el principio una cantidad de memoria bastante grande (lo necesario para un árbol completo de una cierta profundidad que acotaremos).

3.4. Entrada y salida de datos

Para la interfaz de entrada y salida de datos, optamos por el empleo de ficheros de texto simples. En concreto, trabajaremos con un fichero de tareas ("tasks.txt"), que a su vez enlazará, en cada tarea, a un fichero de parámetros. Todos los ficheros de entrada se guardarán en una carpeta `input` al mismo nivel que el ejecutable del programa.

Listado 3.1: tasks.txt

```
1  output: file    // Can be screen, file, text or none
2
3  begin
4
5  name: X000003_bigParameters
6  dataset: 0.982,1172888.982,2345776.982,3518664.982,4691552.982
7  settingsFile: param_set_bigParameters.txt
8
9  name: A000045_fibonacci
10 dataset: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377,
       610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368,
       75025, 121393, 196418, 317811, 514229, 832040, 1346269,
       2178309, 3524578, 5702887, 9227465, 14930352, 24157817
11 settingsFile: param_set_default.txt
12
13 ...
14
15 end
```

En el Listado 3.1 podemos ver un ejemplo de fichero de tareas. La sintaxis específica es:

- Una primera línea `output: type`, donde *type* puede tomar cuatro valores distintos:
  - `screen` indica que queremos salida por pantalla, para estudiar en directo la evolución del algoritmo. Esta salida se compone de texto con los resultados de cada iteración y dos gráficos que se actualizan en tiempo real. Uno es el gráfico de evolución, que estudia el error en función del tiempo, y el otro es una representación de la sucesión estudiada frente a la mejor aproximación de la iteración.
  - `file` indica que la salida se debe volcar a ficheros. La salida se almacenará en una carpeta `results`, y contendrá tres ficheros por cada

secuencia: un log de texto con los resultados de cada iteración y los dos gráficos mencionados en formato `postscript`, con los resultados finales del algoritmo.

- `text` indica salida por pantalla en formato sólo texto, sin gráficos.
- `none` indica que no se debe mostrar nada por pantalla. Este tipo de salida es interesante para propósitos de benchmarking, dado el coste relativamente alto de mostrar por pantalla gráficos y logs.

- Una serie de bloques de tarea, compuestos por:
  - Una línea `name:` *`name`*, donde *name* indicará el nombre de la secuencia a estudiar. Este nombre identificará a la sucesión en los logs, gráficos y ficheros de salida.
  - Una línea `dataset:` , seguida de los elementos de la sucesión a estudiar, separados por comas.
  - Una línea `settingsFile:` *`fichero.txt`*, donde *fichero.txt* será el nombre del fichero de parámetros que queremos emplear en esta secuencia.

En la misma carpeta de entrada `input` se incluirán también los ficheros de parámetros. Estos simplemente serán listados en los que cada línea será del tipo `nombreParametro, t, v`, donde:

- `nombreParametro` indica el nombre del parámetro en cuestión.
- `t` indica el tipo de parámetro:
  - `i` para `int`.
  - `f` para `float`.
  - `i` para `xreal` (véase la Subsección 3.6.2: Types).
  - `s` para `scorereal` (véase la Subsección 3.6.2: Types).
  - `e` para `enumeration` (véase la Subsección 3.6.1: Constants).
- `v` indica el valor del parámetro.

3.5. Parámetros de control

Los parámetros de control que pueden ajustarse son:

- `maxDepth, i`, que controla la máxima profundidad de los árboles de fórmulas (sin contabilizar los nodos terminales).
- `probLeaf, f`, que indica la probabilidad, durante la generación de nuevas fórmulas, de que un nodo sea o no terminal.
- `indInitVal, x`, que indica el valor inicial de las constantes.

- `maxPrev, i`, que indica el máximo número de elementos que las variables hacia atrás podrán utilizar (véase la Subsección 1.6.2: Inclusión de variables temporales).
- `initialTargetLength, i`, que indica la longitud de entrada inicial a considerar. Esto significa que, para una secuencia de longitud 50, por ejemplo, podemos empezar considerando sólo los primeros 15 elementos e ir añadiendo nuevos elementos progresivamente.
- `errorThreshold, s`, que indica el margen de error bajo el cual incrementaremos la longitud considerada de la secuencia de entrada.
- `finishErrorThreshold, s`, que indica el margen de error bajo el cual finalizaremos la ejecución del algoritmo (siempre y cuando la entrada sea ya considerada en su totalidad).
- `errorFunction, e`, que indica la función de error empleada:
  - `kEFuncRel`, para el error relativo.
  - `kEFuncAbs`, para el error absoluto.
- `errorMode, e`, que indica la función de agregación de error empleada:
  - `kErrorMean`, para la media de los errores.
  - `kErrorMax`, para el máximo de los errores.
- `evaluationMode, e`, que indica el método de evaluación empleado:
  - `kEvalSinglePredict`.
  - `kEvalFullSeriesPredict`.

  Ambos métodos se describen en la Subsección 3.6.6.
- `stepUpLag, i`, que indica el mínimo número de iteraciones que deben producirse entre incrementos de la longitud considerada de la secuencia de entrada.
- `populationSize, i`, que indica el tamaño de la población principal para el algoritmo genético.
- `foreignersPercentage, f`, que indica la proporción de individuos de cada nueva generación que se generarán de forma aleatoria.
- `mutationPercentage, f`, que indica la proporción de individuos de cada nueva generación que se generarán por mutación.
- `crossoverPercentage, f`, que indica la proporción de individuos de cada nueva generación que se generarán por cruzamiento.
- `elitistReplicationPercentage, f`, que indica la proporción de individuos de cada nueva generación que se replicarán de forma elitista (copias directas de los mejores individuos de la población).

  No se indica un último parámetro, que corresponderá a los individuos restantes de la población, y que se producirán por replicación no elitista, siguiendo el método del torneo.

- `tournamentSize, i`, que indicará el tamaño de torneo a considerar.
- `numIters, i`, que indicará el número máximo de iteraciones que se ejecutarán para cada secuencia.
- `GPESItersRatio, i`, que indicará cada cuantas iteraciones de Programación Genética realizaremos una iteración de Estrategia Evolutiva.
- `populationSizeES, i`, que indicará el tamaño de la población para la estrategia evolutiva.
- `offspringES, i`, que indicará el número de hijos generados por mutación en la estrategia evolutiva (se mutarán los `offspringES` mejores individuos de la población).
- `numItersES, i`, que indicará el número de iteraciones de Estrategia Evolutiva a realizar en cada ciclo.
- `limitCyclesES, i`, que indicará el número máximo de ciclos consecutivos de Estrategia Evolutiva que un cierto individuo podrá sufrir.
- `coolDownCyclesES, i`, que indicará el número de ciclos de Estrategia Evolutiva que un individuo que haya llegado al límite `limitCyclesES` deberá esperar antes de poder volver al proceso de optimización.

  Estos dos parámetros implican que cada individuo realizará un patrón de ciclos de optimización seguido de unos ciertos ciclos de descanso.
- `initialSigma, x`, que indicará la desviación estándar empleada en la creación de los nuevos individuos de la población de la Estrategia Evolutiva. Este valor será con el que se inicialicen también los parámetros de control de la estrategia evolutiva, y que después irán mutando en cada iteración.
- `minSigma, x`, es el valor mínimo de los parámetros de control de la Estrategia Evolutiva.
- `accurateSizeES, i`, es la cantidad de individuos que se inicializarán con un parámetro de control reducido, de forma que busquen soluciones en un espacio reducido en torno al individuo inicial.
- `coarseSizeES, i`, es la cantidad de individuos que se inicializarán con un parámetro de control incrementado, de forma que busquen soluciones en un espacio ampliado en torno al individuo inicial.

  Este proceso de inicialización fragmentado se describe con más detalle en la Subsección 3.6.11: EvolutionStrategy.

3.6. Estructura del código

El código ha sido descompuesto en diferentes módulos que agrupan las tareas relacionadas.

Inicialmente se planteó el empleo de Programación Orientada a Objetos, sin embargo enseguida se desechó dado el *overhead* o sobrecoste que este paradigma

tiende a acarrear. La descomposición del código en ficheros (la mayor parte asociados a un *namespace* o dominio de nombres independiente) y funciones permite mantener la modularidad deseada y genera un código razonablemente ordenado, eliminando la mayor parte de este *overhead*.

La única excepción se encuentra en la clase Stack, donde implementamos una pila con las distintas operaciones que admitimos en nuestro lenguaje. En este caso, el empleo de una clase nos permite usar limpiamente plantillas (una herramienta de C++ para la reimplementación de código que sólo se distinga en los tipos de datos).

3.6.1. Constants

El fichero de cabecera `constants.h` incluye las constantes generales del algoritmo.

- Alfabeto de fórmula, incluyendo las contantes que asociamos a los elementos del lenguaje, los símbolos de impresión de cada elemento y el número de parámetros que consume cada función. En la práctica sólo empleamos terminales y operadores binarios, pero el código como tal soportaría operadores unarios y sería posible la extensión a operadores de mayor número de operandos (aunque supondría cambiar la forma de almacenar el árbol de la función).
- Constantes para el modo de evaluación. Se verá en la sección dedicada al módulo `Evaluation`.
- Constantes para el modo de cálculo del error. Se verá en la sección dedicada al módulo `Evaluation`.

Listado 3.2: constants.h

```
1  ////////////////////////////////////////////////////////////////////
2  /// File "constants.h"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Author: Daniel Dominguez Catena
5  ////////////////////////////////////////////////////////////////////
6
7  #ifndef GRGA_CONSTANTS_H
8  #define GRGA_CONSTANTS_H
9
10 #include <map>
11
12 namespace GRGA
13 {
14
15   enum eSymbol {
16
17     //Binary Operators
18     kOperatorsBin = 0,
19     kAdd,
20     kSub,
21     kMul,
```

```
22      kDiv,
23      kLog,
24      kPow,
25      kComp,
26      kMod,
27      kOperatorsBinEnd,
28
29      //Operands
30      kOperands = 10,
31      kTime,
32      kPrev,
33      kInd,
34      k1,
35      k0,
36      kneg1,
37      kOperandsEnd,
38
39      //Meta
40      kEnd = 20
41    };
42
43    const std::map<eSymbol, int> mParams = {
44      {kAdd, 1},
45      {kSub, 1},
46      {kMul, 1},
47      {kDiv, 1},
48      {kPow, 1},
49      {kLog, 1},
50      {kComp, 1},
51      {kMod, 1},
52      {kTime, -1},
53      {kPrev, -1},
54      {kInd, -1},
55      {k1, -1},
56      {k0, -1},
57      {kneg1, -1},
58      {kEnd, 1}
59    };
60
61    const std::map<eSymbol, std::string> mStrings = {
62      {kAdd, "+"},
63      {kSub, "-"},
64      {kMul, "*"},
65      {kDiv, "/"},
66      {kPow, "^"},
67      {kLog, "log"},
68      {kComp, ">"},
69      {kMod, "%"},
70      {kTime, "T"},
71      {kPrev, "P"},
72      {kInd, "K"},
73      {k1, "1"},
74      {k0, "0"},
75      {kneg1, "(-1)"},
76      {kEnd, "#"}
77    };
78
79    enum eEvalMode {
80      kEvalSinglePredict,
81      kEvalFullSeriesPredict
82    };
83
```

```
84    enum eErrorMode {
85      kErrorMean,
86      kErrorMax
87    };
88
89    enum eErrorFunction {
90      kEFuncRel,
91      kEFuncAbs
92    };
93
94    const std::map<std::string, int> cStrings = {
95      {"kEvalSinglePredict", kEvalSinglePredict},
96      {"kEvalFullSeriesPredict", kEvalFullSeriesPredict},
97      {"kErrorMean", kErrorMean},
98      {"kErrorMax", kErrorMax},
99      {"kEFuncRel", kEFuncRel},
100     {"kEFuncAbs", kEFuncAbs},
101   };
102 }
103
104 #endif // GRGA_CONSTANTS_H
```

3.6.2. Types

El fichero de cabecera `types.h` contiene la descripción de los tipos que empleamos durante la ejecución del algoritmo.

- Definición del tipo `xreal`. Este es el tipo que emplearemos para las operaciones internas en la fórmula y para los datos de entrada. En general usaremos double.
- Definición del tipo `scorereal`. Este es el tipo que emplearemos para almacenar el error calculado en los resultados. En general usaremos double.

La idea de tener estos dos tipos separados es poder implementar con un real los resultados y con un entero las operaciones binarias, en caso de que consideremos que la función objetivo sólo hará uso de este tipo (series como la sucesión de Fibonacci, por ejemplo). En la práctica hemos mantenido siempre estos dos a double, al encontrarnos con que los problemas que sólo hacían uso de enteros eran lo suficientemente simples como para resolverse rápidamente manteniendo un tipo real para operaciones internas.

- Tipo `fType`, para el almacenamiento de las fórmulas. Contiene:
  - Un puntero a enteros sin signo `expression`, que señala el inicio del array que contiene los símbolos de la fórmula en sí.
  - Un puntero a enteros sin signo `prev`, que señala el inicio del array donde se almacenan los índices de retroceso de las variables referentes a elementos anteriores de la secuencia.
    - Esto es, si en la posición 6 del array de fórmula hemos colocado un símbolo `kPrev` que debería hacer referencia al elemento de la

secuencia de entrada anterior, en este array colocaremos un 1. Si debe hacer referencia al elemento anterior al anterior, colocaremos un 2, etc.

- Un puntero a `xreal parameters`, que señala el inicio del array donde almacenaremos los parámetros reales correspondientes a los símbolos de tipo constante del array de fórmula.
  - Este array sólo se emplea para las constantes de tipo `kInd`, que son las que derivaremos. Las constantes 0 y 1 no serán estables en el tiempo y no se derivarán.
  - Se guardará el valor del parámetro en la misma posición que el símbolo en el array de fórmula.
- Un entero `length`, que indica la longitud de la fórmula.
- Un entero `maxLength`, que indica la longitud máxima de la fórmula (tamaño de memoria reservado). En la práctica no lo empleamos, al tener todos los individuos la misma longitud, pero en caso de cambiar la representación de la fórmula será de gran utilidad.
- Un entero `n`, que indica el número de constantes en la fórmula generada. Este número se usa en la estrategia evolutiva, para evitar realizar iteraciones inútiles para fórmulas que no tengan constantes.

- Tipo `iType`, para el almacenamiento de los individuos de la población. Contiene:
  - Un puntero `formula` a `fType`, que hace referencia a la fórmula del individuo.
  - Un `scorereal score`, que almacenará el valor de la función de fitness del individuo.
  - Un entero `age`, que emplearemos para indicar la edad (en generaciones) del individuo. Este valor se empleará para ajustar los parámetros de la estrategia evolutiva.
  - Los enteros `ESCycles` y `ESCoolDown`, que nos servirán para manejar la frecuencia con la que un individuo pasa por estrategia evolutiva.
    - `ESCycles` indicará el número total de pasadas de estrategia evolutiva que ha sufrido un individuo.
    - `ESCoolDown` nos servirá para controlar que un individuo que haya realizado un cierto número de pasadas de estrategia evolutiva espere un cierto tiempo antes de volver a pasar por estrategia evolutiva.

- Tipo `pType`, que almacenará una población de individuos.
  - Se implementa como un `std::vector` de punteros a `iType`. En este caso el *overhead* del vector de C++ es mínimo, y a cambio ganamos en facilidad para introducir y eliminar rápidamente individuos de la población y para ordenarlos por fitness.

- Tipo `iESType`, que almacenará los individuos durante la fase de estrategia evolutiva. Contiene:

- Un puntero `formula` a `fType`, que referencia a la fórmula específica del individuo.
- Un `scorereal score`, que almacenará el valor de la función de fitness del individuo.
- Un puntero a `xreal parameters`, que señala el inicio del array donde almacenaremos los parámetros reales correspondientes a los símbolos de tipo constante del array de fórmula.
  - Este array será la variación del homónimo de la fórmula en cuestión que lleva asociado el individuo.
- Un puntero a `xreal control`, que llevará los parámetros de control propios de la estrategia evolutiva.

- Tipo `pESType`, que almacenará la población de individuos de la estrategia evolutiva.
  - Internamente se implementa como un `std::list` de punteros a `iESType`. Al no necesitar acceso por índice a elementos arbitrarios (por el funcionamiento interno del algoritmo podemos emplear un iterador sin problemas), podemos utilizar un `std::list` en vez de un `std::vector`, como se utilizó para `pType`. El hecho de emplear una lista en vez de un vector será práctico en nuestro caso, al disponer `std::list` de un método para la mezcla, del que haremos uso.

- Tipo `dType`, que emplearemos para almacenar los datos de la secuencia de entrada. Se compone de:
  - Un puntero a `xreal data`, que marcará el principio de una secuencia de `xreal` con los elementos de la sucesión a estudiar.
  - Un entero `length` que nos dará la longitud de la secuencia de entrada.
  - Un `std::string name` donde almacenaremos el nombre de la secuencia (para luego poner los nombres correctos en los gráficos y ficheros generados, principalmente).

Además, se implementan en este fichero varias funciones `inline` auxiliares, para borrar los contenidos de los distintos tipos de datos, comparaciones y ordenaciones.

Listado 3.3: types.h

```
1  ////////////////////////////////////////////////////////////////////
2  /// File "types.h"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Author: Daniel Dominguez Catena
5  ////////////////////////////////////////////////////////////////////
6
7  #ifndef GRGA_TYPES_H
8  #define GRGA_TYPES_H
9
10 #include <list>
```

```
11 #include <vector>
12
13 namespace GRGA
14 {
15
16   #ifndef XREAL
17   #define XREAL
18   //typedef float xreal;
19   //#define gpc_plot_2dx gpc_plot_2df
20   typedef double xreal;
21   #define gpc_plot_2dx gpc_plot_2dd
22   #endif
23
24   #ifndef SCOREREAL
25   #define SCOREREAL
26   //typedef float scorereal;
27   //#define gpc_plot_2ds gpc_plot_2df
28   typedef double scorereal;
29   #define gpc_plot_2ds gpc_plot_2dd
30   #endif
31
32   //Formula
33   typedef struct
34   {
35     unsigned int * expression;
36     unsigned int * prev;
37     xreal * parameters;
38     int length;
39     int n = -1;
40     int maxLength;
41   } fType;
42
43   //GP Individual
44   typedef struct
45   {
46     fType * formula;
47     scorereal score = -1;
48     int age = 1;
49     int ESCycles = 0;
50     int ESCoolDown = 0;
51   } iType;
52
53   //GP Population
54   typedef std::vector<iType *> pType;
55
56
57   //ES Individual
58   typedef struct
59   {
60     fType * formula;
61     xreal * parameters;
62     xreal * control;
63     scorereal score = -1;
64   } iESType;
65
66   //ES Population
67   typedef std::list<iESType *> pESType;
68
69   //Data
70   typedef struct
71   {
72     xreal * data;
```

```
73      int length;
74      std::string name;
75    } dType;
76
77  }
78  #endif // GRGA_TYPES_H
```

3.6.3. Main

En el fichero `main.cpp` leemos un fichero de tareas (`tasks.txt`) donde se detallan las funciones a ajustar, así como el fichero de parámetros que se debe emplear para cada una de ellas, y lanzamos la ejecución de cada tarea individual.

- Desde este módulo se llama al módulo `Parameters` para realizar la carga de los parámetros de cada tarea.
- Además, se deriva la ejecución de cada tarea individual al módulo `Direction`, que orquesta la ejecución del algoritmo en sí desde su método `run`.

Listado 3.4: main.cpp

```
1  ////////////////////////////////////////////////////////////////////////
2  /// File "main.cpp"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Author: Daniel Dominguez Catena
5  ////////////////////////////////////////////////////////////////////////
6
7  #include <fstream>
8  #include <iostream>
9  #include <sstream>
10 #include <string>
11 #include "auxFuncs.h"
12 #include "direction.h"
13 #include "omp.h"
14 #include "parameters.h"
15 #include "types.h"
16
17 using namespace std;
18 using namespace GRGA;
19
20 typedef struct inputData {
21   std::vector<xreal> data;
22   std::string name;
23 } inputData;
24
25 int main()
26 {
27
28   initRand();
29
30   ifstream infile("./input/tasks.txt");
31   string line;
```

```
32
33    if (!infile.is_open())
34      {
35        cout << "Error reading task file" << endl;
36        exit(-1);
37      }
38
39    getline(infile, line);
40
41    {
42      istringstream iss(line);
43      string output;
44
45      iss.ignore(50, ':');
46      if (iss.peek() == ' ')
47        iss.ignore(1, ' ');
48
49      iss >> output;
50
51      if (output.compare("screen") == 0)
52        {
53          cout << "Full output on screen" << endl << endl;
54          GRGA_PLOT_RESULTS = 1;
55          GRGA_PLOT_RESULTS_TO_FILE = 0;
56          LOG_TO_FILE = 0;
57          OMIT_LOG = 0;
58        }
59      else if (output.compare("file") == 0)
60        {
61          cout << "Full output on files" << endl << endl;
62          GRGA_PLOT_RESULTS = 0;
63          GRGA_PLOT_RESULTS_TO_FILE = 1;
64          LOG_TO_FILE = 1;
65          OMIT_LOG = 0;
66        }
67      else if (output.compare("text") == 0)
68        {
69          cout << "Text output on screen" << endl << endl;
70          GRGA_PLOT_RESULTS = 0;
71          GRGA_PLOT_RESULTS_TO_FILE = 0;
72          LOG_TO_FILE = 0;
73          OMIT_LOG = 0;
74        }
75      else if (output.compare("none") == 0)
76        {
77          cout << "No output" << endl << endl;
78          GRGA_PLOT_RESULTS = 0;
79          GRGA_PLOT_RESULTS_TO_FILE = 0;
80          LOG_TO_FILE = 0;
81          OMIT_LOG = 1;
82        }
83      else
84        {
85          cout << "Unrecognised output type" << endl << endl;
86          exit(-1);
87        }
88    }
89
90    while (getline(infile, line) && (line.compare(0, 5, "begin") !=
        0)) {}
91
92    if (line.compare(0, 5, "begin") != 0)
```

```
93      {
94        cout << "Error reading task file" << endl;
95        cout << "No begin found on task file" << endl;
96        exit(-1);
97      }
98
99    while (getline(infile, line) &&
100          ((line.length() == 0) ||
101          (line.front() == '%'))) {}
102
103   while (line.compare(0, 3, "end") != 0)
104     {
105       // Read task
106       istringstream iss(line);
107       string name;
108       std::vector<xreal> data;
109
110       iss.ignore(50, ':');
111       if (iss.peek() == ' ')
112         iss.ignore(1, ' ');
113
114       iss >> name;
115
116       while (getline(infile, line) &&
117              ((line.length() == 0) ||
118              (line.front() == '%'))) {}
119
120       iss.str(line);
121       iss.clear();
122
123       iss.ignore(50, ':');
124
125       xreal tmp;
126
127       while (iss >> tmp)
128         {
129           data.push_back(tmp);
130
131           if (iss.peek() == ',')
132             iss.ignore(1, ',');
133           iss >> ws;
134         }
135
136
137       while (getline(infile, line) &&
138              ((line.length() == 0) ||
139              (line.front() == '%'))) {}
140
141       iss.str(line);
142       iss.clear();
143       iss.ignore(50, ':');
144       string parametersFile;
145       iss >> parametersFile;
146
147       parametersFile = "./input/" + parametersFile;
148
149       cout << "Round begin: " << endl;
150       cout << "Target data: " << name << endl;
151
152       // Load data
153       dType * dS = new dType();
154       dS->data = &(data[0]);
```

```
155       dS->length = data.size();
156       dS->name = name;
157       targetData = dS;
158
159       Parameters::loadParametersFromFile(parametersFile);
160
161       if (LOG_TO_FILE == 1)
162         {
163           //Redirect output
164           std::string fileName = "./results/" + targetData->name +
    "_log.txt";
165           freopen (fileName.c_str(), "w", stdout);
166         }
167
168       // Run
169       double start = omp_get_wtime();
170
171       Direction::run();
172       double stop = omp_get_wtime();
173
174       if (LOG_TO_FILE == 1)
175         {
176           cout << "Ran in " << stop - start << " seconds" << endl;
177           cout << "for target data: " << name << endl << endl;
178
179           // Restore output
180           fflush(stdout);
181           freopen ("/dev/tty", "a", stdout);
182         }
183
184       cout << "Ran in " << stop - start << " seconds" << endl;
185       cout << "for target data: " << name << endl << endl;
186
187       while (getline(infile, line) &&
188              ((line.length() == 0) ||
189              (line.front() == '%'))) {}
190
191     }
192
193
194   return 0;
195
196 }
```

3.6.4. Parameters

Los parámetros para la ejecución del algoritmo se guardan en un `std::map` para posibilitar la modificación sobre la marcha de los distintos valores, así como la carga inicial.

Este mapa asociará las claves, que serán `std::string` a los valores de los parámetros. Para poder emplear un mapa único para los parámetros de distintos tipos (`float`, `int`, `xreal`...), emplearemos un `union` de C++, una estructura capaz de contener un elemento de una serie de tipos diferente.

Para la carga de los parámetros en sí, disponemos de dos funciones:

- `loadParametersFromFile(std::string fileName)` carga los parámetros que se indican en el fichero `filename`.
- `generateDerivedParameters()` obtiene algunos parámetros derivados del resto. Esta función se llama justo después de lanzar la anterior.

Listado 3.5: parameters.h

```
1  //////////////////////////////////////////////////////////////////////
2  /// File "parameters.h"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Author: Daniel Dominguez Catena
5  //////////////////////////////////////////////////////////////////////
6
7  #ifndef GRGA_PARAMETERS_H
8  #define GRGA_PARAMETERS_H
9
10 #include <map>
11 #include "constants.h"
12 #include "types.h"
13
14 namespace GRGA
15 {
16   extern int GRGA_PLOT_RESULTS;
17   extern int GRGA_PLOT_RESULTS_TO_FILE;
18
19   extern int LOG_TO_FILE;
20   extern int OMIT_LOG;
21
22   typedef union {
23     float f;
24     xreal x;
25     scorereal s;
26     int i;
27   } parameter;
28
29   extern std::map<std::string, parameter> p;
30
31   namespace Parameters
32   {
33     void loadParametersFromFile(std::string fileName);
34     void generateDerivedParameters();
35   }
36
37   extern dType * targetData;
38
39
40 }
41
42 #endif // GRGA_PARAMETERS_H
```

Listado 3.6: parameters.cpp

```
1  //////////////////////////////////////////////////////////////////////
2  /// File "parameters.cpp"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
```

```
4  /// Author: Daniel Dominguez Catena
5  ////////////////////////////////////////////////////////////////////
6
7  #include <fstream>
8  #include <iostream>
9  #include <sstream>
10 #include "parameters.h"
11
12 using namespace GRGA;
13 using namespace std;
14
15 int GRGA::GRGA_PLOT_RESULTS = 0;
16 int GRGA::GRGA_PLOT_RESULTS_TO_FILE = 0;
17
18 int GRGA::LOG_TO_FILE = 0;
19 int GRGA::OMIT_LOG = 0;
20
21 map<string, parameter> GRGA::p;
22
23 void Parameters::loadParametersFromFile(string fileName)
24 {
25   ifstream infile(fileName);
26   string line;
27
28   if (!infile.is_open())
29     {
30       cout << "Error opening parameter file ";
31       cout << "[" << fileName << "]" << endl;
32     }
33
34   while (getline(infile, line))
35     {
36       istringstream iss(line);
37       string name;
38       char type;
39       xreal x;
40       scorereal s;
41       float f;
42       int i;
43       string e;
44
45       getline(iss, name, ',');
46
47       if ((name.front() == '%') || (name.length() >= line.length())
     )
48         continue;
49
50       if (iss.peek() == ',')
51           iss.ignore(1, ',');
52       if (iss.peek() == ' ')
53           iss.ignore(1, ' ');
54       iss >> type;
55
56       if (iss.peek() == ',')
57           iss.ignore(1, ',');
58       if (iss.peek() == ' ')
59           iss.ignore(1, ' ');
60
61       switch (type)
62         {
63         case 'i':
```

```
64            iss >> i;
65            p[name].i = i;
66            break;
67          case 'x':
68            iss >> x;
69            p[name].x = x;
70            break;
71          case 'f':
72            iss >> f;
73            p[name].f = f;
74            break;
75          case 's':
76            iss >> s;
77            p[name].s = s;
78            break;
79          case 'e':
80            iss >> e;
81            i = cStrings.at(e);
82            p[name].i = i;
83            break;
84          default:
85            cout << "Unrecognized type error: " << type << endl;
86            exit(-1);
87          }
88      }
89
90      generateDerivedParameters();
91  }
92
93  void Parameters::generateDerivedParameters()
94  {
95    p["maxFLength"].i = 2 << p["maxDepth"].i;
96    p["initialTargetLength"].i = min(p["initialTargetLength"].i,
97        targetData->length);
98    p["currentTargetLength"].i = p["initialTargetLength"].i;
99
100   p["foreignersSize"].i = p["populationSize"].i *
101       p["foreignersPercentage"].f;          // New individuals
102   p["mutationSize"].i = p["populationSize"].i *
103       p["mutationPercentage"].f;            // Mutated individuals
104   p["crossoverSize"].i = p["populationSize"].i *
105       p["crossoverPercentage"].f;           // Breeded individuals
106   p["elitistReplicationSize"].i = p["populationSize"].i *
107       p["elitistReplicationPercentage"].f;
108 }
109
110 dType * GRGA::targetData = NULL;
```

3.6.5. Direction

Este módulo recoge el núcleo de ejecución tanto de la sección de Programación Genética como de la de Estrategia Evolutiva.

El punto de entrada para la ejecución es la función `run()`.

- Primero llamaremos a `GeneticProgramming::initialPopulation()` para generar la población inicial sobre la que trabajaremos.

- Exigimos que cada iteración empiece con una población completamente evaluada y ordenada, por lo que realizaremos la pasada de evaluación sobre la población inicial (mediante `Evaluation::score()`) y ordenaremos. La ordenación se realiza en una de las funciones auxiliares definidas en

- Iteramos sobre el número de iteraciones especificado en el parámetro `numIters`.

    - En cada iteración derivamos la siguiente población mediante una llamada a `deriveNextGeneration(pType * population)`.
    - La población derivada debe ser evaluada antes de la siguiente iteración para facilitar el trabajo de las funciones de torneo. En nuestro caso, aprovecharemos este requerimiento para ejecutar la parte de Estrategia Evolutiva.
        - Para cada individuo de la población lanzaremos la ejecución de la Estrategia Evolutiva para mejorar sus parámetros constantes. Esto se realiza en la función `runES(iType *f)`. Como efecto secundario, a la salida de esta función tendremos el individuo ya evaluado.
        - Existe un parámetro `GPESitersRatio` que permite evitar que la Estrategia Evolutiva se ejecute en todas las iteraciones. La idea es darle algo más de tiempo a la Programación Genética para hacer su trabajo y derivar fórmulas plausibles antes de optimizarlas por completo. Esto significa que las iteraciones en las que no ejecutemos la Estrategia Evolutiva debemos forzar la evaluación de los individuos.
        - Esta parte se ejecuta en paralelo mediante un `#pragma omp`. Esto agiliza enormemente la ejecución, al ser el punto de computación más intensiva del algoritmo.
    - Una vez evaluada, realizamos la ordenación de la población en base a su fitness (en nuestro caso el `score` que obtuvimos en la evaluación).
    - Consideramos como resultado de la iteración el mejor individuo de ésta, fácilmente localizable al quedar en la última posición de la población ordenada.
    - Si corresponde, mostramos por pantalla los resultados.
    - A partir del resultado de la iteración, decidimos si debemos incrementar el tamaño de la muestra sobre la que estamos trabajando. En nuestra implementación empezaremos a trabajar sobre los primeros `initialTargetLength` elementos de la serie objetivo, y cada vez que el mejor individuo de la iteración baje de un cierto umbral de error `errorThreshold`, incrementaremos el tamaño de la muestra.
        - En caso de incrementarlo, deberemos reevaluar todos los individuos de la población.
    - Si al finalizar la iteración el mejor individuo obtiene un error menor que un cierto umbral `finishErrorThreshold`, finalizaremos la ejecución del algoritmo y devolveremos éste individuo como resultado.

- Si al finalizar todas las iteraciones no hemos obtenido un resultado definitivo (por debajo del umbral de error exigido), devolveremos el mejor resultado de la última iteración.

El cálculo de la siguiente población se obtiene en la función `deriveNextGeneration(pType * population)`:

- Empleamos una serie de parámetros `*Size` para regular los distintos métodos de obtener individuos nuevos:
  - Reservamos espacio para una población de `populationSize` elementos.
  - Introducimos `foreignersSize` individuos completamente aleatorios.
  - Introducimos `crossoverSize` individuos por cruzamiento de padres. Para el cruzamiento, se eligen los dos progenitore por el método del torneo.
  - Introducimos `mutationSize` individuos por mutación de un padre. Elegimos al padre por torneo.
  - Forzamos la replicación exacta de los `elitistReplicationSize` mejores individuos de la población.
  - Los restantes individuos hasta completar el tamaño de población requerido los obtenemos por replicación de los individuos de la población, eligiendo al azar los progenitores.

Por lo que respecta a la parte de Estrategia Evolutiva, se realiza en la función `iType * Direction::runES(iType * i)`:

- Consideramos los parámetros de `coolDownCyclesES` y `limitCyclesES` para decidir si el individuo debe pasar por Estrategia Evolutiva o si este turno debe saltárselo (al haber realizado ya muchos seguidos).
- Revisamos el número de parámetros que tiene la fórmula a evaluar. Si la fórmula no tiene parámetros, no tiene sentido ejecutar la Estrategia Evolutiva.
- A través de `pESType * EvolutionStrategy::initialPopulation(fType *f, int age)` generaremos la población inicial de individuos para Estrategia Evolutiva.
- Evaluamos y ordenamos la población.
- Después iteramos para el número `numItersES` de iteraciones.
  - Aprovechando que la población ya está ordenada, crearemos copias mutadas de los `offspringES` mejores individuos de la población. La mutación la realizaremos a través de la función `iESType * EvolutionStrategy::mutate(iESType * p1)`.
  - Evaluaremos y ordenaremos estos individuos y los mezclaremos con la población original.

- Recortaremos esta población para la siguiente iteración, eliminando los peores hasta volver al tamaño de población original.

- Una vez ejecutadas todas las iteraciones, seleccionaremos el mejor individuo de la población, y transplantaremos sus parámetros y función de fitness (error en evaluación) al individuo original.

Además, disponemos de un par de funciones para mostrar los gráficos de los resultados, a través de una librería externa sobre Gnuplot.

Listado 3.7: direction.h

```
1  ////////////////////////////////////////////////////////////////////
2  /// File "direction.h"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Author: Daniel Dominguez Catena
5  ////////////////////////////////////////////////////////////////////
6
7  #ifndef GRGA_DIRECTION_H
8  #define GRGA_DIRECTION_ H
9
10 #include "gnuplot_c.h"
11 #include "types.h"
12
13 namespace GRGA
14 {
15
16   namespace Direction
17   {
18     static h_GPC_Plot *h2DPlot = NULL;
19
20     scorereal run();
21     pType * deriveNextGeneration(pType *population);
22     iType * runES(iType * f);
23     void interruptHandler(int s);
24   }
25
26 }
27
28 #endif // GRGA_DIRECTION_H
```

Listado 3.8: direction.cpp

```
1  ////////////////////////////////////////////////////////////////////
2  /// File "direction.cpp"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Author: Daniel Dominguez Catena
5  ////////////////////////////////////////////////////////////////////
6
7  #include <cmath>
8  #include <csignal>
9  #include <cstdio>
10 #include <ctime>
11 #include <iostream>
```

```
12 #include <sstream>
13 #include <stdlib.h>
14 #include "auxFuncs.h"
15 #include "constants.h"
16 #include "direction.h"
17 #include "es.h"
18 #include "evaluation.h"
19 #include "generation.h"
20 #include "gp.h"
21 #include "omp.h"
22 #include "parameters.h"
23 #include "plot.h"
24 #include "tournament.h"
25
26 using namespace GRGA;
27
28 scorereal Direction::run()
29 {
30
31   struct sigaction sigIntHandler;
32
33   sigIntHandler.sa_handler = interruptHandler;
34   sigemptyset(&sigIntHandler.sa_mask);
35   sigIntHandler.sa_flags = 0;
36
37   sigaction(SIGINT, &sigIntHandler, NULL);
38
39   scorereal scores[p["numIters"].i];
40   scorereal lengths[p["numIters"].i];
41   scorereal thresholdBar[] = { p["finishErrorThreshold"].s, p["
       finishErrorThreshold"].s };
42
43   if (GRGA_PLOT_RESULTS == 1)
44     {
45       Plot::initializePlots();
46     }
47
48   auto population = GeneticProgramming::initialPopulation();
49
50   for(auto it = population->begin(); it != population->end(); it++)
51     {
52       (* it)->score = Evaluation::score((* it)->formula);
53     }
54
55   sortPType(population);
56
57   int stepUpCounter = 0;
58
59   int i;
60   for (i = 0; i < p["numIters"].i; i++)
61     {
62       // The population begins the iteration already ordered,
63       // worst to best.
64
65       population = deriveNextGeneration(population);
66
67       #pragma omp parallel for num_threads(omp_get_num_procs())
       shared(population)
68       for(int e = 0; e < p["populationSize"].i; e++)
69         {
70           iType * originalElem = population->at(e);
71           if ((i % p["GPESItersRatio"].i == 0) || (i == p["numIters
```

```
    "].i - 1))
72              {
73                iType * newElem = runES(originalElem);
74                if (newElem != originalElem)
75                  {
76                    clearIType(originalElem);
77                    population->at(e) = newElem;
78                  }
79                originalElem = population->at(e);
80              }
81            if (originalElem->score == -1)
82              {
83                originalElem->score = Evaluation::score(originalElem
    ->formula);
84              }
85            originalElem->age++;
86          }
87
88        sortPType(population);
89
90        auto bestind = population->back();
91
92        if (!OMIT_LOG)
93          {
94            cout << "Best individual [it " << i << "] for dataset \""
     << targetData->name << "\": " << endl;
95            printFormula(bestind->formula);
96            cout << "Score: " << bestind->score << endl;
97            cout << "Age: " << bestind->age << " ESCycles: " <<
    bestind->ESCycles <<
98                 " ESCoolDown: " << bestind->ESCoolDown << endl;
99            cout << "Current lenght = " << p["currentTargetLength"].i
     << "/" << targetData->length << endl << endl;
100
101         }
102
103       scores[i] = bestind->score;
104       lengths[i] = p["currentTargetLength"].i;
105
106
107       if (GRGA_PLOT_RESULTS == 1)
108         {
109           if (i % 10 == 0 || i == p["numIters"].i - 1 ||
110               bestind->score < p["finishErrorThreshold"].s)
111             {
112               Plot::plotIteration(i, scores, thresholdBar, lengths,
     bestind);
113             }
114
115         }
116
117       stepUpCounter++;
118       if ((stepUpCounter > p["stepUpLag"].i) && (bestind->score <=
    p["errorThreshold"].s))
119         {
120           if (p["currentTargetLength"].i < targetData->length)
121             {
122               cout << "That was enough! Step it up a bit!" << endl;
123               p["currentTargetLength"].i++;
124               cout << "Current lenght = " << p["currentTargetLength
    "].i <<
125                    "/" << targetData->length << endl << endl;
```

```
126               stepUpCounter = 0;
127
128               #pragma omp parallel for num_threads(
    omp_get_num_procs()) shared(population)
129               for(int e = 0; e < p["populationSize"].i; e++)
130                 {
131                   iType * elem = population->at(e);
132                   elem->age = 0;
133                   elem->ESCoolDown = 0;
134                   elem->ESCycles = 0;
135                   elem->score = Evaluation::score(elem->formula);
136                 }
137             }
138           else
139             {
140               if (bestind->score < p["finishErrorThreshold"].s)
141                 {
142                   cout << "Looks like we have a winner!" << endl;
143                   break;
144                 }
145             }
146         }
147
148     }
149
150   auto bestind = population->back();
151
152   if (!OMIT_LOG)
153     {
154       cout << "Final best individual: " << endl;
155       printFormula(bestind->formula);
156       cout << "Score: " << bestind->score << " evaluated over a
    length of " <<
157              p["currentTargetLength"].i << " out of " << targetData
    ->length << endl << endl;
158     }
159
160   scorereal finalDeviation = bestind->score;
161
162   if(GRGA_PLOT_RESULTS_TO_FILE == 1)
163     {
164       Plot::plotFinalResults(i, scores, thresholdBar, lengths,
    bestind);
165     }
166
167
168   clearPType(population);
169
170   if (GRGA_PLOT_RESULTS == 1 )
171     {
172       printf ("\nHit <CR> to continue ....\n"); getchar (); // Wait
     for <CR>
173       Plot::closePlots();
174     }
175
176   return finalDeviation;
177 }
178
179 pType * Direction::deriveNextGeneration(pType * population)
180 {
181   auto newPopulation = new pType;
182   newPopulation->reserve(p["populationSize"].i);
```

```
183
184   int e = 0;
185
186   for (e = 0; e < p["foreignersSize"].i; e++)
187     {
188       auto childf = Generation::generate();
189
190       auto child = new iType;
191       child->formula = childf;
192       child->score = -1;
193
194       newPopulation->push_back(child);
195     }
196
197   for (e = 0; e < p["crossoverSize"].i; e++)
198     {
199       auto parenti1 = Tournament::tournament(population);
200       auto parenti2 = Tournament::tournament(population);
201       auto parentf1 = population->at(parenti1)->formula;
202       auto parentf2 = population->at(parenti2)->formula;
203       auto childf = GeneticProgramming::crossover(parentf1,
      parentf2);
204       auto child = new iType;
205       child->formula = childf;
206       child->score = -1;
207
208       newPopulation->push_back(child);
209     }
210
211   for (e = 0; e < p["mutationSize"].i; e++)
212     {
213       auto parenti1 = Tournament::tournament(population);
214       auto parentf1 = population->at(parenti1)->formula;
215       auto childf = GeneticProgramming::mutate(parentf1);
216
217       auto child = new iType;
218       child->formula = childf;
219       child->score = -1;
220
221       newPopulation->push_back(child);
222     }
223
224   for (e = 0; e < p["elitistReplicationSize"].i; e++)
225     {
226       newPopulation->push_back(population->back());
227       population->pop_back();
228     }
229
230   for (e = 0; (int) newPopulation->size() < p["populationSize"].i;
      e++)
231     {
232       int childIndex = RANDOM(0, (int) (population->size() - 1));
233
234       newPopulation->push_back(population->at(childIndex));
235
236       population->at(childIndex) = population->back();
237       population->pop_back();
238     }
239
240   clearPType(population);
241   return newPopulation;
242 }
```

```
243
244 iType * Direction::runES(iType * i)
245 {
246   if (i->ESCycles > p["limitCyclesES"].i && i->ESCoolDown < p["
      coolDownCyclesES"].i)
247     {
248       i->ESCoolDown++;
249       if (i->ESCoolDown >= p["coolDownCyclesES"].i)
250         i->ESCycles = 0;
251       return i;
252     }
253   i->ESCycles++;
254   i->ESCoolDown = 0;
255
256   unsigned int * cur = i->formula->expression;
257
258   if (i->formula->n == -1)
259     {
260       i->formula->n = 0;
261
262       while (*cur != kEnd)
263         {
264           if (*cur == kInd)
265             {
266               i->formula->n += 1;
267             }
268           cur++;
269         }
270     }
271
272   if (i->formula->n == 0)
273     {
274       return i;
275     }
276
277   pESType * population = EvolutionStrategy::initialPopulation(i->
      formula, i->age);
278   pESType::iterator it;
279
280   for ( it = population->begin(); it != population->end(); ++it )
281     {
282       (*it)->score = Evaluation::score((*it));
283     }
284   population->sort(compareIESType);
285
286   float lastSuccessfulMutationProportion = 1.0/5.0;
287
288   for (int i = 0; i < p["numItersES"].i; i++)
289     {
290       pESType * children = new pESType();
291       it = population->begin();
292
293       int successfulMutations = 0;
294
295       while ((it != population->end()) && ((int) children->size() <
       p["offspringES"].i))
296         {
297           auto child = EvolutionStrategy::mutate(* it,
      lastSuccessfulMutationProportion);
298           child->score = Evaluation::score(child);
299           children->push_back(child);
300
```

```
301             if (child->score < (* it)->score)
302               {
303                 successfulMutations++;
304               }
305           }
306
307         lastSuccessfulMutationProportion = (float)
        successfulMutations / p["offspringES"].i;
308
309         children->sort(compareIESType);
310
311         population->merge((* children), compareIESType);
312
313         while ((int) population->size() > p["populationSizeES"].i)
314           {
315             clearIESType( population->back() );
316             population->pop_back();
317           }
318
319         delete children;
320
321       }
322
323     while( population->size() > 1 )
324       {
325         clearIESType( population->back() );
326         population->pop_back();
327       }
328
329     iESType * best = population->front();
330     delete population;
331
332     fType * fS = new fType();
333     fS->expression = (unsigned int *)malloc(sizeof(unsigned int) * p
        ["maxFLength"].i);
334     fS->prev = (unsigned int *)malloc(sizeof(unsigned int) * p["
        maxFLength"].i);;
335     fS->parameters = best->parameters;
336     fS->length = best->formula->length;
337     fS->n = best->formula->n;
338     fS->maxLength = p["maxFLength"].i;
339
340     for (int i = 0; i < fS->length; i++)
341       {
342         fS->expression[i] = best->formula->expression[i];
343         fS->prev[i] = best->formula->prev[i];
344       }
345
346     iType * result = new iType();
347     result->formula = fS;
348     result->score = best->score;
349     result->age = i->age;
350     result->ESCoolDown = i->ESCoolDown;
351     result->ESCycles = i->ESCycles;
352
353     free(best->control);
354     delete best;
355
356     return result;
357 }
358
359 void Direction::interruptHandler (int s)
```

```
360 {
361   printf("Caught signal %d\n",s);
362
363   if (GRGA_PLOT_RESULTS == 1)
364     {
365       printf ("\nHit <CR> to continue ....\n"); getchar (); // Wait
       for <CR>
366     }
367   if (GRGA_PLOT_RESULTS == 1 || GRGA_PLOT_RESULTS_TO_FILE == 1)
368     {
369       Plot::closePlots();
370     }
371
372   exit(1);
373 }
```

3.6.6. Evaluation

Éste módulo se ocupa de la evaluación de fórmulas (hipótesis) sobre la serie objetivo.

En la función `score(fType * formula)` obtenemos el error cometido por la función en el fragmento de serie que estamos estudiando en el momento (que puede estar limitado por el parámetro `currentTargetLength`):

- Evaluamos la función desde el elemento `maxPrev` de la serie hasta el `currentTargetLength`.
  - Debemos limitar los primeros elementos para evitar que en las fórmulas que hagan uso de referencias a los elementos anteriores estos queden sin evaluar.
  - Limitamos los últimos para facilitar que la función primero adquiera las características notables de la serie objetivo y sólo después las generalice para el resto de la serie.
  - La evaluación podrá realizarse de dos formas distintas según el valor del parámetro `evaluationMode`:
    - Si toma el valor de la constante `kEvalSinglePredict` se evaluará para cada elemento de la serie de forma independiente. Esto significa que para cada elemento consideraremos como elementos previos los de la serie original.
    - Si, por el contrario, toma el valor de `kEvalFullSeriesPredict`, la serie se evaluará como un conjunto. Esto significa que para cada elemento consideraremos como previos los anteriores resultados calculados con la fórmula.
- Obtendremos una expresión agregada del error cometido a partir de los errores individuales.
  - Se puede controlar mediante el parámetro `errorFunction` si se empleará el error relativo (`kEFuncRel`) o el error absoluto (`kEFuncAbs`) como medidas del error. En el caso del error relativo, se medirá en tanto por uno respecto del valor esperado.

    - En función del valor del parámetro `errorMode` empleamos bien la media (`kErrorMean`) o el máximo (`kErrorMax`) como función de agregación.

Las funciones `evaluate` y `evaluateFullSeries` realizarán la evaluación para un elemento concreto de la serie.

- Gracias a la representación que empleamos para la fórmula, simplemente recorreremos la fórmula por completo y le pasaremos cada símbolo a un objeto `Stack`, que realizará el cálculo típico de notación postfija.
    - El trabajo que se realizará en estas funciones será la sustitución de todas los terminales (constantes o variables) numéricos por su valor específico.
- El último símbolo de todos, `kEnd`, se corresponderá con el de la clase `Stack` que devuelve el resultado del cálculo. Retornaremos este valor como salida de la función para esta posición de la serie a evaluar.

Listado 3.9: evaluation.h

```
1  ////////////////////////////////////////////////////////////////////////
2  /// File "es.h"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Author: Daniel Dominguez Catena
5  ////////////////////////////////////////////////////////////////////////
6
7  #ifndef GRGA_EVALUATION_H
8  #define GRGA_EVALUATION_H
9
10 #include "gnuplot_c.h"
11 #include "types.h"
12
13 namespace GRGA
14 {
15
16   namespace Evaluation
17   {
18     extern h_GPC_Plot *h2DPlot;
19
20     scorereal score(fType * formula, bool print = false);
21     inline scorereal score(iESType * i)
22     {
23       xreal * tmp = i->formula->parameters;
24       i->formula->parameters = i->parameters;
25       scorereal res = score(i->formula);
26       i->formula->parameters = tmp;
27       return res;
28     }
29
30     xreal evaluate(fType * formula, unsigned int index);
31     xreal evaluateFullSeries(fType * formula, unsigned int index,
32                              xreal * prevs, int prevIndex);
33   }
```

```
34
35 }
36
37 #endif // GRGA_EVALUATION_H
```

Listado 3.10: evaluation.cpp

```
1  ////////////////////////////////////////////////////////////////////////
2  /// File "evaluation.cpp"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Author: Daniel Dominguez Catena
5  ////////////////////////////////////////////////////////////////////////
6
7  #include <cmath>
8  #include <limits>
9  #include "auxFuncs.h"
10 #include "constants.h"
11 #include "evaluation.h"
12 #include "stack.h"
13 #include "plot.h"
14
15 using namespace GRGA;
16 using namespace std;
17
18 h_GPC_Plot * Evaluation::h2DPlot = NULL;
19 scorereal Evaluation::score(fType * formula, bool print)
20 {
21   int maxPrev = p["maxPrev"].i;
22   int errorFunc = p["errorFunction"].i;
23
24   xreal results[p["currentTargetLength"].i - maxPrev];
25   scorereal error = 0;
26   if (p["evaluationMode"].i == kEvalSinglePredict)
27     {
28       for (int i = maxPrev; i < p["currentTargetLength"].i; i++)
29         {
30           xreal result = evaluate(formula, i);
31           xreal expected = targetData->data[i];
32
33           scorereal localError;
34           if (errorFunc == kEFuncRel)
35             {
36               if (expected == 0)
37                 localError = fabs(result);
38               else
39                 localError = fabs((result - expected) / expected);
40             }
41           else
42             {
43               localError = fabs(result - expected);
44             }
45
46           if (p["errorMode"].i == kErrorMean)
47             error += localError;
48           else
49             error = max(error, localError);
50
51           results[i - maxPrev] = result;
52         }
```

```
53        error = error / p["currentTargetLength"].i;
54
55      }
56    else
57      {
58        xreal previous[maxPrev];
59        int pIndex = 0;
60        for (int i = 0; i < maxPrev; i++)
61          previous[i] = targetData->data[i];
62
63        int errorMode = p["errorMode"].i;
64
65        for (int i = maxPrev; i < p["currentTargetLength"].i; i++)
66          {
67            xreal result = evaluateFullSeries(formula, i, previous,
      pIndex);
68            xreal expected = targetData->data[i];
69
70            scorereal localError;
71            if (errorFunc == kEFuncRel)
72              {
73                if (expected == 0)
74                  localError = fabs(result);
75                else
76                  localError = fabs((result - expected) / expected);
77              }
78            else
79              {
80                localError = fabs(result - expected);
81              }
82
83            if (errorMode == kErrorMean)
84              error += localError;
85            else
86              error = max(error, localError);
87
88            if (maxPrev > 0)
89              {
90                previous[pIndex] = result;
91                pIndex++;
92                pIndex %= maxPrev;
93              }
94
95            results[i - maxPrev] = result;
96          }
97        if (errorMode == kErrorMean)
98          error = error / p["currentTargetLength"].i;
99      }
100
101
102   if (print)
103     {
104       Plot::plotEvaluation(results);
105     }
106
107   if ( std::isinf( error ))
108   {
109     return std::numeric_limits<double>::max();
110   }
111   if ( std::isnan( error ))
112   {
113     return std::numeric_limits<double>::max();
```

```
114   }
115   return error;
116 }
117 
118 xreal Evaluation::evaluate(fType * formula, unsigned int index)
119 {
120   auto stack = new Stack<xreal>;
121   eSymbol s;
122   int time = 0;
123   unsigned int * fCursor = formula->expression;
124   xreal * vCursor= formula->parameters;
125   unsigned int * pCursor= formula->prev;
126 
127   while (*fCursor != kEnd)
128     {
129       s = static_cast<eSymbol>(*fCursor);
130       switch (s)
131         {
132         case kInd:
133           stack->ind(*vCursor);
134           break;
135         case kPrev:
136           stack->ind(targetData->data[index - *pCursor]);
137           break;
138         case kTime:
139           stack->ind(index);
140           break;
141         case k1:
142           stack->ind(1.0);
143           break;
144         case k0:
145           stack->ind(0.0);
146           break;
147         case kneg1:
148           stack->ind(-1.0);
149           break;
150         default:
151           stack->operate(s);
152           break;
153           break;
154         }
155       fCursor++;
156       pCursor++;
157       vCursor++;
158       time++;
159     }
160   xreal result = stack->end();
161 
162   delete stack;
163   return result;
164 }
165 
166 xreal Evaluation::evaluateFullSeries(fType * formula, unsigned int
      index,
167                                      xreal * prevs, int prevIndex)
168 {
169   auto stack = new Stack<xreal>;
170   eSymbol s;
171   int time = 0;
172   unsigned int * fCursor = formula->expression;
173   xreal * vCursor= formula->parameters;
174   unsigned int * pCursor= formula->prev;
```

```
175
176   while (*fCursor != kEnd)
177     {
178       s = static_cast<eSymbol>(*fCursor);
179       switch (s)
180         {
181         case kInd:
182           stack->ind(*vCursor);
183           break;
184         case kPrev:
185           stack->ind(prevs[ (prevIndex + p["maxPrev"].i -
186               *pCursor) % p["maxPrev"].i ]);
187           break;
188         case kTime:
189           stack->ind(index);
190           break;
191         case k1:
192           stack->ind(1.0);
193           break;
194         case k0:
195           stack->ind(0.0);
196           break;
197         case kneg1:
198           stack->ind(-1.0);
199           break;
200         default:
201           stack->operate(s);
202           break;
203           break;
204         }
205       fCursor++;
206       pCursor++;
207       vCursor++;
208       time++;
209     }
210   xreal result = stack->end();
211
212   delete stack;
213   return result;
214 }
```

3.6.7. Stack

La clase `Stack` implementa una pila de reales y una serie de operaciones sobre ellos.

Para el almacenamiento de los valores reales, empleamos un `std::stack`, por sencillez y velocidad.

La clase se implementa empleando las plantillas o templates propios de C++, lo cual permite que trabaje con diferente tipos de datos sin problemas. El motivo de utilizar plantillas en vez del typedef que ya se utiliza en el resto del código es que en caso de emplear tipos "exóticos" (complejos, enteros, etc.) la implementación cambiaría substancialmente, a diferencia del resto de módulos. Al emplear templates, podrían coexistir implementaciones de la clase para diferentes tipos sin mayor problema.

La clase implementa tres métodos simples como interfaz hacia el exterior:

- `void operate(eSymbol s)`, que realiza la operación indicada por el símbolo en cuestión.
  - Este método actúa como proxy para el resto de métodos de operación.
  - Los métodos de operación son pequeños fragmentos de código que recogen una serie de operandos de la parte alta de la pila, operan sobre ellos, y hacen *push* del resultado de vuelta a la pila.
- `void ind(type f)`, que empuja un operando numérico a la pila.
- `type end()`, que recoge el resultado final de la pila y lo devuelve.

Listado 3.11: stack.h

```
1  ////////////////////////////////////////////////////////////////////////
2  /// File "stack.h"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Author: Daniel Dominguez Catena
5  ////////////////////////////////////////////////////////////////////////
6
7  #ifndef GRGA_STACK_H
8  #define GRGA_STACK_H
9
10 #include <stack>
11 #include "constants.h"
12
13 namespace GRGA
14 {
15   template <typename type>
16   class Stack
17   {
18   private:
19     std::stack<type> * stack;
20   public:
21     ~Stack();
22     Stack();
23     void operate(eSymbol s);
24     void ind(type f);
25     type end();
26   private:
27     void add();
28     void sub();
29     void mul();
30     void div();
31     void pow();
32     void log();
33     void comp();
34     void mod();
35   };
36
37 }
38
39 #endif // GRGA_STACK_H
```

Listado 3.12: stack.cpp

```
1  ////////////////////////////////////////////////////////////////////
2  /// File "stack.cpp"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Author: Daniel Dominguez Catena
5  ////////////////////////////////////////////////////////////////////
6
7  #include <cmath>
8  #include <iostream>
9  #include "stack.h"
10
11 using namespace GRGA;
12
13 template <typename type>
14 Stack<type>::Stack()
15 {
16   stack = new std::stack<type>();
17 }
18
19 template <typename type>
20 Stack<type>::~Stack()
21 {
22   delete(stack);
23 }
24
25 template <typename type>
26 void Stack<type>::operate(eSymbol s)
27 {
28   switch (s)
29     {
30     case kAdd:
31       add();
32       break;
33     case kSub:
34       sub();
35       break;
36     case kMul:
37       mul();
38       break;
39     case kDiv:
40       div();
41       break;
42     case kPow:
43       pow();
44       break;
45     case kLog:
46       log();
47       break;
48     case kComp:
49       comp();
50       break;
51     case kMod:
52       mod();
53       break;
54     default:
55       std::cout << "Unrecognised operand" << std::endl;
56       exit(-1);
57     }
58 }
59
```

```
60  template <typename type>
61  void Stack<type>::add()
62  {
63    type op1 = stack->top();
64    stack->pop();
65    type op2 = stack->top();
66    stack->pop();
67    stack->push(op1 + op2);
68  }
69
70  template <typename type>
71  void Stack<type>::sub()
72  {
73    type op1 = stack->top();
74    stack->pop();
75    type op2 = stack->top();
76    stack->pop();
77    stack->push(op1 - op2);
78  }
79
80  template <typename type>
81  void Stack<type>::mul()
82  {
83    type op1 = stack->top();
84    stack->pop();
85    type op2 = stack->top();
86    stack->pop();
87    stack->push(op1 * op2);
88  }
89
90  template <typename type>
91  void Stack<type>::div()
92  {
93    type op1 = stack->top();
94    stack->pop();
95    type op2 = stack->top();
96    stack->pop();
97    if (op2 == 0)
98      stack->push(op1 == 0);
99    else if (op1 == 0)
100     stack->push(1.0);
101   else
102     stack->push(op1 / op2);
103 }
104
105 template <typename type>
106 void Stack<type>::pow()
107 {
108   type op1 = stack->top();
109   stack->pop();
110   type op2 = stack->top();
111   stack->pop();
112   type res = std::pow(op1, op2);
113   if ( std::isnan( res ))
114     stack->push(0);
115   else
116     stack->push(res);
117 }
118
119 template <typename type>
120 void Stack<type>::log()
121 {
```

```
122   type op1 = stack->top();
123   stack->pop();
124   type op2 = stack->top();
125   stack->pop();
126   type res = std::log2(op1) / std::log2(op2);
127   if ( std::isnan( res ))
128     stack->push(0);
129   else
130     stack->push(res);
131 }
132
133 template <typename type>
134 void Stack<type>::comp()
135 {
136   type op1 = stack->top();
137   stack->pop();
138   type op2 = stack->top();
139   stack->pop();
140   stack->push((op1 > op2) ? 1.0 : 0.0);
141 }
142
143 template <typename type>
144 void Stack<type>::mod()
145 {
146   type op1 = floor(stack->top());
147   stack->pop();
148   type op2 = floor(stack->top());
149   stack->pop();
150   if (op2 <= 0 || std::isinf( op2 ) || std::isinf( op1 ))
151     {
152       stack->push(0.0);
153     }
154   else
155     {
156       type res = (int) op1 % (int) op2;
157       res = (res < 0) ? res + op2 : res;
158       stack->push(res);
159     }
160 }
161
162 template <typename type>
163 void Stack<type>::ind(type f)
164 {
165   stack->push(f);
166 }
167
168 template <typename type>
169 type Stack<type>::end()
170 {
171   type ret = stack->top();
172   stack->pop();
173   if (!stack->empty())
174     {
175       std::cout << "Formula finished with " << stack->size() << "
      items left" << std::endl;
176       exit(-1);
177     }
178
179   if ( std::isinf( ret ))
180     return 0;
181
182   if ( std::isnan( ret ))
```

```
183     return 0;
184
185   return ret;
186 }
187
188 template class Stack<float>;
189 template class Stack<double>;
190 template class Stack<int>;
```

3.6.8. Generation

En el módulo `Generate` implementamos la generación de nuevas funciones aleatorias.

Existe un único método, `fType * generate()`, que devuelve un puntero a una nueva fórmula generada. Para generar la fórmula, seguimos el siguiente método:

- Reservar espacio para los distintos arrays que componen la fórmula. Es necesario reservar un array auxiliar por cada array del resultado, ya que desconocemos la longitud final de la fórmula que vamos a generar y tenemos que generarla desde el nodo raíz (que en nuestra implementación quedará al final del array).
- Crearemos una pila auxiliar que nos permitirá contabilizar el número de hijos que faltan por añadir para un cierto nivel.
    - Como inicialmente necesitamos al menos 1 elemento en nuestra fórmula, colocaremos un 1 en la parte superior de la pila.
- Iteraremos mientras el stack no esté vacío (mientras la fórmula no esté completa).
    - Decidiremos si el siguiente elemento será o no una hoja, en función de un parámetro `probLeaf` y de la profundidad del árbol, forzando un elemento hoja siguiendo la probabilidad indicada o siempre que estemos en el último nivel permitido (según el parámetro `maxDepth`.
    - Generaremos un nodo al azar, sea terminal (hoja) u operador, e indicaremos en el stack cuántos operandos necesitamos.
        - En el caso de las hojas, indicaremos un 0, ya que no requieren operandos por debajo suya.
        - En el resto de casos, indicaremos un 2, ya que nuestra implementación sólo considera operadores binarios, de momento.
    - Subiremos por la pila, saltando todos los niveles completos (que se corresponderán con una cabeza de pila igual a 0), hasta volver al último nivel incompleto.
- Por último, invertiremos los vectores auxiliares y crearemos la fórmula a devolver.

Este recorrido de la pila se corresponde con un recorrido en profundidad, y supone una implementación relativamente simple y que garantiza árboles por debajo de la profundidad especificada, con una cierta variedad de formas y tamaños.

Listado 3.13: generation.h

```
1  ////////////////////////////////////////////////////////////////////////
2  /// File "generation.h"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Author: Daniel Dominguez Catena
5  ////////////////////////////////////////////////////////////////////////
6
7  #ifndef GRGA_RANDOMFUNCTION_H
8  #define GRGA_RANDOMFUNCTION_H
9
10 #include "types.h"
11
12 namespace GRGA
13 {
14
15   namespace Generation
16   {
17     fType * generate();
18   }
19
20 }
21
22 #endif // GRGA_RANDOMFUNCTION_H
```

Listado 3.14: generation.cpp

```
1  ////////////////////////////////////////////////////////////////////////
2  /// File "generation.cpp"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Author: Daniel Dominguez Catena
5  ////////////////////////////////////////////////////////////////////////
6
7  #include <random>
8  #include <stack>
9  #include "auxFuncs.h"
10 #include "constants.h"
11 #include "generation.h"
12 #include "parameters.h"
13
14 using namespace GRGA;
15
16 fType * Generation::generate()
17 {
18   unsigned int * auxf = (unsigned int *)malloc(sizeof(unsigned int)
       * p["maxFLength"].i);
19   int auxfIndex = 0;
20   unsigned int * f = (unsigned int *)malloc(sizeof(unsigned int) *
       p["maxFLength"].i);
21   int fIndex = 0;
```

```
22  unsigned int * prev = (unsigned int *)malloc(sizeof(unsigned int)
      * p["maxFLength"].i);
23  unsigned int * auxp = (unsigned int *)malloc(sizeof(unsigned int)
      * p["maxFLength"].i);
24  xreal * var = (xreal *)malloc(sizeof(xreal) * p["maxFLength"].i);
25  xreal * auxv = (xreal *)malloc(sizeof(xreal) * p["maxFLength"].i)
    ;
26
27  int lvl = 0;
28  std::stack<int> * stack = new std::stack<int>();
29  stack->push(1);
30
31  bool leaf;
32
33  while (!stack->empty())
34    {
35      leaf = ((xreal) rand() / RAND_MAX) < p["probLeaf"].f;
36      leaf |= ( lvl >= p["maxDepth"].i );
37      stack->top()--;
38      lvl++;
39      if (leaf)
40        {
41          auxf[auxfIndex] = RANDOMEXCLUSIVE(kOperands, kOperandsEnd
    );
42          stack->push(0);
43          switch (auxf[auxfIndex]) {
44            case kInd:
45              auxv[auxfIndex] = p["indInitVal"].x;
46              break;
47            case kPrev:
48              if (p["maxPrev"].i < 1)
49                auxf[auxfIndex] = kTime;
50              else
51                auxp[auxfIndex] = RANDOM(1, p["maxPrev"].i);
52              break;
53            default:
54              break;
55            }
56        }
57      else
58        {
59          auxf[auxfIndex] = RANDOMEXCLUSIVE(kOperatorsBin,
    kOperatorsBinEnd);
60          stack->push(2);
61        }
62
63      while(!stack->empty() && stack->top() == 0)
64        {
65          stack->pop();
66          lvl--;
67        }
68      auxfIndex++;
69    }
70
71  auxfIndex--;
72  while (auxfIndex >= 0)
73    {
74      f[fIndex] = auxf[auxfIndex];
75      var[fIndex] = auxv[auxfIndex];
76      prev[fIndex] = auxp[auxfIndex];
77
78      auxfIndex--;
```

```
79        fIndex++;
80      }
81    f[fIndex] = kEnd;
82    fIndex++;
83
84
85    fType * fS = new fType();
86
87    fS->expression = f;
88    fS->prev = prev;
89    fS->parameters = var;
90    fS->length = fIndex;
91    fS->maxLength = p["maxFLength"].i;
92
93    free(auxf);
94    free(auxv);
95    free(auxp);
96    delete stack;
97
98    return fS;
99  }
```

3.6.9. Tournament

Al trabajar con poblaciones completamente ordenadas, la implementación del típico método del torneo es notablemente simple. En nuestro caso, los detalles del método de torneo desarrollado son:

- Determinista, en el sentido de que elegimos siempre al mejor de los candidatos.
- Con reemplazamiento, lo cual significa que no limitamos la reaparición de individuos varias veces en el mismo torneo.

A nivel de implementación, manejamos un único método `int tournament(pType * population)` que recibe la población y devuelve el índice del individuo ganador del torneo. La obtención de este ganador es tan simple como generar `tournamentSize` índices, y devolver el mayor de ellos.

Listado 3.15: tournament.h

```
1  ////////////////////////////////////////////////////////////////////
2  /// File "tournaments.h"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Author: Daniel Dominguez Catena
5  ////////////////////////////////////////////////////////////////////
6
7  #ifndef GRGA_TOURNAMENT_H
8  #define GRGA_TOURNAMENT_H
9
10 #include "types.h"
11
12 namespace GRGA
```

```
13 {
14   namespace Tournament
15   {
16     // Returns the index of the winner
17     int tournament(pType * population);
18   }
19 }
20
21 #endif // GRGA_TOURNAMENT_H
```

Listado 3.16: tournament.cpp

```
1  ////////////////////////////////////////////////////////////////////////
2  /// File "tournament.cpp"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Author: Daniel Dominguez Catena
5  ////////////////////////////////////////////////////////////////////////
6
7  #include "auxFuncs.h"
8  #include "tournament.h"
9
10 using namespace GRGA;
11
12 int Tournament::tournament(pType * population)
13 {
14   int winner = 0;
15
16   for (int i = 0; i < p["tournamentSize"].i; i++)
17     {
18       int candidate = RANDOM(0, (int) population->size() - 1);
19       if (candidate > winner)
20         {
21           winner = candidate;
22         }
23     }
24   return winner;
25 }
```

3.6.10. GeneticProgramming

El módulo `GeneticProgramming` se ocupa de las tres componentes específicas a la parte de programación genética: el cruzamiento, la mutación y la generación de la población inicial.

La función `fType * crossover(fType * p1, fType * p2)` cruza dos fórmulas para obtener una hija. Nuestra implementación es la habitual para árboles, el sobrecruzamiento de subárboles. Esto implica seleccionar un nodo de cada uno de los dos padres e intercambiar los subárboles correspondientes:

- Primero reservamos espacio para los arrays necesarios.
    - De la misma forma que en la generación de individuos aleatorios, trabajaremos desde el final hacia adelante, de forma que necesitaremos arrays auxiliares para luego invertir el resultado.

- Generamos los índices de los nodos a cruzar. Para esto se emplea la función `int AuxFuncs::getRandomNode(fType * f)`, que una vez generado un índice al azar en el rango de la longitud del array de la fórmula, avanza hasta el nodo no terminal más próximo.
- Crearemos el hijo en tres fases:
  - Primero, avanzaremos el primer padre hasta su índice de corte.
    - En concreto, haremos el recorrido copiando a la vez sus elementos al hijo. Además, almacenaremos en una pila la información de su estructura, de forma que sepamos en qué nivel nos encontramos al trasplantar el subárbol del segundo padre.
    - También haremos saltar el puntero de posición del primer padre hasta justo después del subárbol que vamos a podar. En la tercera fase necesitaremos tener el puntero en la posición correcta.
  - Después, copiaremos el subárbol del segundo padre al hijo.
    - Para cada elemento a copiar, primero comprobaremos si por profundidad debe ser terminal. Si debería ser terminal y no lo es, generaremos un terminal al azar. En cualquier otro caso, copiaremos.
  - Por último, replicaremos el fragmento restante del primer padre, si lo hubiese.
- Después invertiremos los arrays auxiliares y devolveremos el hijo en sí.

Para la mutación (función `fType * mutate(fType * p1)`), implementamos una de las más simples, mutando contra una nueva fórmula aleatoria. >Temporal, si tengo tiempo implementaré alguna mutación más eficiente y "normal". Acabo de recordar que lo dejé como opción temporal y nunca llegué a implementarla.

Por último, la generación de una población inicial (método `pType * initialPopulation()`) es directa, generando los individuos necesarios mediante la función `Generation::generate()`.

Listado 3.17: gp.h

```
1  ////////////////////////////////////////////////////////////////////
2  /// File "gp.h"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Author: Daniel Dominguez Catena
5  ////////////////////////////////////////////////////////////////////
6
7  #ifndef GENETICPROGRAMMING_H
8  #define GENETICPROGRAMMING_H
9
10 #include "types.h"
11
12 namespace GRGA
13 {
14   namespace GeneticProgramming
15   {
```

```
16      fType * crossover(fType * p1, fType * p2);
17      fType * mutateByRandomCrossover(fType * p1);
18      fType * mutate(fType * p1);
19      pType * initialPopulation();
20    }
21  }
22
23  #endif // MUTATOR_H
```

Listado 3.18: gp.cpp

```
1   ////////////////////////////////////////////////////////////////////////
2   /// File "gp.cpp"
3   /// Project GRGA: Generalized Regression based on Genetic
        Algorithms
4   /// Author: Daniel Dominguez Catena
5   ////////////////////////////////////////////////////////////////////////
6
7   #include <stack>
8   #include <string.h>
9   #include "auxFuncs.h"
10  #include "constants.h"
11  #include "generation.h"
12  #include "gp.h"
13  #include "parameters.h"
14
15  using namespace std;
16  using namespace GRGA;
17
18  fType * GeneticProgramming::crossover(fType * p1, fType * p2)
19  {
20    int maxFLength = p["maxFLength"].i;
21    int maxDepth = p["maxDepth"].i;
22    xreal indInitVal = p["indInitVal"].x;
23    int maxPrev = p["maxPrev"].i;
24
25    unsigned int * auxf = (unsigned int *)malloc(sizeof(unsigned int)
        * maxFLength);
26    int auxfIndex = 0;
27    unsigned int * f = (unsigned int *)malloc(sizeof(unsigned int) *
      maxFLength);
28    int fIndex = 0;
29    unsigned int * prev = (unsigned int *)malloc(sizeof(unsigned int)
        * maxFLength);
30    unsigned int * auxp = (unsigned int *)malloc(sizeof(unsigned int)
        * maxFLength);
31    xreal * var = (xreal *)malloc(sizeof(xreal) * maxFLength);
32    xreal * auxv = (xreal *)malloc(sizeof(xreal) * maxFLength);
33
34    int p1fIndex = getRandomNode(p1);
35    int p1auxfIndex = p1->length - 2;
36    int p2fIndex = getRandomNode(p2);
37
38    int lvl = 0;
39    std::stack<int> * stack = new std::stack<int>();
40    stack->push(1);
41
42    bool leaf;
43
44    eSymbol s;
```

```
45
46    while (!stack->empty() && p1auxfIndex != p1fIndex)
47      {
48        auxf[auxfIndex] = p1->expression[p1auxfIndex];
49        auxp[auxfIndex] = p1->prev[p1auxfIndex];
50        auxv[auxfIndex] = p1->parameters[p1auxfIndex];
51
52        stack->top()--;
53        lvl++;
54
55        s = static_cast<eSymbol>(auxf[auxfIndex]);
56        stack->push(mParams.at(s) + 1);
57
58        while(!stack->empty() && stack->top() == 0)
59          {
60            stack->pop();
61            lvl--;
62          }
63
64        auxfIndex++;
65        p1auxfIndex--;
66      }
67
68    int stackCounter = 1;
69
70    while (stackCounter > 0 && p1fIndex >= 0)
71      {
72        s = static_cast<eSymbol>(p1->expression[p1fIndex]);
73
74        stackCounter += mParams.at(s);
75        p1fIndex--;
76      }
77
78    stackCounter = 1;
79    bool finished = false;
80    int params;
81
82    while ((!stack->empty()) && (stackCounter > 0) && !finished)
83      {
84        stack->top()--;
85        lvl++;
86
87        leaf = lvl - 1 >= maxDepth;
88        leaf = leaf && isOperator(p2->expression[p2fIndex]);
89
90        if (leaf)
91          {
92            auxf[auxfIndex] = RANDOMEXCLUSIVE(kOperands, kOperandsEnd
      );
93            stack->push(0);
94            switch (auxf[auxfIndex])
95              {
96              case kInd:
97                auxv[auxfIndex] = indInitVal;
98                break;
99              case kPrev:
100               if (maxPrev < 1)
101                 auxf[auxfIndex] = kInd;
102               else
103                 auxp[auxfIndex] = RANDOM(1, maxPrev);
104               break;
105             default:
```

```
106               break;
107             }
108
109           p2fIndex--;
110           int skipStackCounter = 1;
111           while (skipStackCounter > 0)
112             {
113               s = static_cast<eSymbol>(p2->expression[p2fIndex]);
114               skipStackCounter += mParams.at(s);
115               p2fIndex--;
116             }
117
118           s = static_cast<eSymbol>(auxf[auxfIndex]);
119           params = mParams.at(s);
120           stackCounter += params;
121
122         }
123       else
124         {
125           auxf[auxfIndex] = p2->expression[p2fIndex];
126           auxp[auxfIndex] = p2->prev[p2fIndex];
127           auxv[auxfIndex] = p2->parameters[p2fIndex];
128
129           s = static_cast<eSymbol>(auxf[auxfIndex]);
130           params = mParams.at(s);
131           stack->push(params + 1);
132
133           stackCounter += params;
134         }
135
136       auxfIndex++;
137       p2fIndex--;
138
139       if(stackCounter == 0)
140         {
141           finished = true;
142           break;
143         }
144
145       while(!stack->empty() && stack->top() == 0)
146         {
147           stack->pop();
148           lvl--;
149         }
150
151     }
152
153   while (p1fIndex >= 0)
154     {
155       auxf[auxfIndex] = p1->expression[p1fIndex];
156       auxp[auxfIndex] = p1->prev[p1fIndex];
157       auxv[auxfIndex] = p1->parameters[p1fIndex];
158
159       auxfIndex++;
160       p1fIndex--;
161     }
162   auxfIndex--;
163   while (auxfIndex >= 0)
164     {
165       f[fIndex] = auxf[auxfIndex];
166       var[fIndex] = auxv[auxfIndex];
167       prev[fIndex] = auxp[auxfIndex];
```

```
168
169       auxfIndex--;
170       fIndex++;
171     }
172   f[fIndex] = kEnd;
173   fIndex++;
174
175   fType * fS = new fType();
176
177   fS->expression = f;
178   fS->prev = prev;
179   fS->parameters = var;
180   fS->length = fIndex;
181   fS->maxLength = maxFLength;
182
183   free(auxf);
184   free(auxv);
185   free(auxp);
186   delete stack;
187
188   return fS;
189 }
190
191
192 fType * GeneticProgramming::mutateByRandomCrossover(fType * p1)
193 {
194   fType * p2 = Generation::generate();
195   fType * c = crossover(p1, p2);
196
197   clearFType(p2);
198   return c;
199 }
200
201 fType * GeneticProgramming::mutate(fType * p1)
202 {
203   auto child = new fType;
204   int length = p1->length;
205
206   child->expression = (unsigned int *)malloc(sizeof(unsigned int) *
        p["maxFLength"].i);
207   memcpy(child->expression, p1->expression, length * sizeof(
      unsigned int));
208   child->prev = (unsigned int *)malloc(sizeof(unsigned int) * p["
      maxFLength"].i);
209   memcpy(child->prev, p1->prev, length * sizeof(unsigned int));
210   child->parameters = (xreal *)malloc(sizeof(xreal) * p["maxFLength
      "].i);
211   memcpy(child->parameters, p1->parameters, length * sizeof(xreal))
      ;
212
213   child->length = length;
214   child->maxLength = p1->maxLength;
215
216   int mutationPoint = RANDOM(0, length - 2);
217
218   eSymbol s = static_cast<eSymbol> (child->expression[mutationPoint
      ]);
219   if ((s > kOperatorsBin) && (s < kOperatorsBinEnd))
220     {
221       child->expression[mutationPoint] = RANDOMEXCLUSIVE(
      kOperatorsBin, kOperatorsBinEnd);
222     }
```

```
223   else if ((s > kOperands) && (s < kOperandsEnd))
224     {
225       s = static_cast<eSymbol> (RANDOMEXCLUSIVE(kOperands,
      kOperandsEnd));
226       child->expression[mutationPoint] = s;
227       switch (s) {
228         case kInd:
229           child->parameters[mutationPoint] = p["indInitVal"].x;
230           break;
231         case kPrev:
232           if (p["maxPrev"].i < 1)
233             child->expression[mutationPoint] = kTime;
234           else
235             child->prev[mutationPoint] = RANDOM(1, p["maxPrev"].i);
236           break;
237         default:
238           break;
239         }
240     }
241
242   return child;
243
244 }
245
246
247 pType * GeneticProgramming::initialPopulation()
248 {
249   auto pop = new pType;
250   pop->reserve(p["populationSize"].i);
251
252   for (int i = 0; i < p["populationSize"].i; i++)
253     {
254       auto individual = new iType;
255       individual->formula = Generation::generate();
256       individual->score = -1;
257       pop->push_back(individual);
258     }
259
260   return pop;
261 }
```

3.6.11. EvolutionStrategy

El módulo `EvolutionStrategy` se ocupa de los fragmentos específicos de la otra gran parte del algoritmo, la Estrategia Evolutiva: la mutación y la generación de la población inicial.

En nuestro caso, hemos optado por una estrategia evolutiva de tipo $(\mu+\lambda)-ES$. Esto significa que manejaremos una población inicial de μ individuos y en cada iteración generaremos λ descendientes por mutación. Después, el total de estos individuos competirán y sobrevivirán los μ más aptos de todos.

Para la representación emplearemos un esquema de n variables de control σ_k, una para cada uno de los n parámetros x_k de la fórmula de entrada, ignorando la posible correlación entre ellas.

Para la mutación (función `iESType * mutate(iESType * p1)`), empleamos el algoritmo clásico:

- Calcular los dos parámetros de aprendizaje τ (local) y τ' (global) como
  - $\tau' = \frac{1}{\sqrt{2 \times n}}$
  - $\tau = \frac{1}{\sqrt{2 \times \sqrt{n}}}$
- Mutar los parámetros de control del individuo siguiendo:
  - $\sigma_i = \sigma_i \times \exp(\tau' \times N(0,1) + \tau \times N_i(0,1))$
- Mutar los parámetros objetivo del individuo siguiendo:
  - $x'_i = x_i + \sigma'_i \times N_i(0,1)$

Para la generación de la población inicial (función `pESType * initialPopulation(fType *f, int age)`) generaremos los individuos a partir del individuo original, desviándonos de éste según una distribución normal independiente para cada parámetro, siempre con media el parámetros en cuestión y una cierta desviación típica variable. En concreto:

- El primer individuo será siempre una copia del padre original en cuanto a variables, y con sus parámetros de control inicializados a $initialSigma/age$.
  - Esto implica reducir el sigma inicial para afinar la búsqueda. Conforme más antiguo es el individuo, más podremos acotarla a su entorno cercano.
- Los siguientes `accurateSizeES` individuos emplearán una variable $\sigma = initialSigma/(\log(age) + 1)$
  - Los parámetros objetivos se obtendrán con una distribución normal de media el parámetro original y desviación típica el parámetro σ.
  - Los parámetros de control se inicializarán directamente a σ.
  - Estos individuos se centrarán en la zona cercana al individuo original, tanto más cuanto más viejo sea.
- Los siguientes `coarseSize` individuos harán lo mismo con $\sigma = initialSigma \times (\log(age) + 1)$.
- Estos individuos se alejarán de zona cercana al individuo original, tanto más cuanto más viejo sea.
- Los restantes individuos emplearán $\sigma = initialSigma$.
- Estos individuos se encontrarán siempre en una región razonable alrededor del individuo original, de forma constante pese al envejecimiento del individuo en cuestión.

La motivación para realizar esta subdivisión inicial reside en el hecho de que realizaremos la Estrategia Evolutiva de forma "segmentada", no en una pasada constante si no eliminando la población entre ciclos (alternándose con las iteraciones de Programación Genética). Es por esto que conviene mantener algo de información, para facilitar al algoritmo que en las pocas iteraciones que vaya a tener disponibles encuentre una mejora respecto del individuo original.

Listado 3.19: es.h

```
1  ////////////////////////////////////////////////////////////////////
2  /// File "es.h"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Author: Daniel Dominguez Catena
5  ////////////////////////////////////////////////////////////////////
6
7  #ifndef EVOLUTIONSTRATEGY_H
8  #define EVOLUTIONSTRATEGY_H
9
10 #include "types.h"
11
12 namespace GRGA
13 {
14   namespace EvolutionStrategy
15   {
16     iESType * mutate(iESType * p1, float proportion);
17     pESType * initialPopulation(fType * f, int age);
18   }
19 }
20
21 #endif // EVOLUTIONSTRATEGY_H
```

Listado 3.20: es.cpp

```
1  ////////////////////////////////////////////////////////////////////

2  /// File "es.cpp"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Author: Daniel Dominguez Catena
5  ////////////////////////////////////////////////////////////////////


6
7  #include <cmath>
8  #include <iostream>
9  #include <stack>
10 #include "auxFuncs.h"
11 #include "constants.h"
12 #include "es.h"
13 #include "parameters.h"
14
15 using namespace std;
16 using namespace GRGA;
17
18 iESType * EvolutionStrategy::mutate(iESType * p1, float proportion
       )
19 {
20   auto individual = new iESType;
21   individual->formula = p1->formula;
22   individual->score = -1;
23   individual->parameters = (xreal *)malloc(sizeof(xreal) * p1->
       formula->length);
24   individual->control = (xreal *)malloc(sizeof(xreal) * p1->formula
       ->length);
25
26   int n = p1->formula->n;
27   float correctionFactor;
28   if (proportion <= 0.2)
29     correctionFactor = 10;
30   else
```

```
31      correctionFactor = 1;
32    xreal tau_ = correctionFactor * 1.0 / sqrt(2.0  * n); // Global
        learning rate
33    xreal tau = correctionFactor * 1.0 / sqrt(2.0  * sqrt(n)); //
        Local learning rate
34
35    xreal n01global = dist01(gen);
36    xreal n01local;
37    xreal minSigma = p["minSigma"].x;
38
39    for (int i = 0; i < p1->formula->length; i++)
40      {
41        if (p1->formula->expression[i] == kInd)
42          {
43              n01local = dist01(gen);
44              individual->control[i] = max(p1->control[i] * std::exp(
        tau_ * n01global + tau * n01local), minSigma);
45              individual->parameters[i] = p1->parameters[i] +
        individual->control[i] * n01local;
46          }
47      }
48
49    return individual;
50  }
51
52  pESType * EvolutionStrategy::initialPopulation(fType *f, int age)
53  {
54    auto pop = new pESType;
55
56    auto individual = new iESType;
57    individual->formula = f;
58    individual->score = -1;
59
60    individual->parameters = (xreal *)malloc(sizeof(xreal) * f->
        length);
61    individual->control = (xreal *)malloc(sizeof(xreal) * f->length);
62
63    for (int j = 0; j < f->length; j++)
64      {
65        individual->parameters[j] = f->parameters[j];
66        individual->control[j] = p["initialSigma"].x / (std::log2(age
        ) + 1);
67      }
68
69    pop->push_back(individual);
70
71    int i;
72
73    xreal initialSigma = p["initialSigma"].x;
74    int accurateSizeES = p["accurateSizeES"].i;
75    int coarseSizeES = p["coarseSizeES"].i;
76
77    xreal denominator = age + 1;
78
79    for (i = 1; i < p["populationSizeES"].i; i++)
80      {
81        auto individual = new iESType;
82        individual->formula = f;
83        individual->score = -1;
84
85        individual->parameters = (xreal *)malloc(sizeof(xreal) * f->
        length);
```

```
86        individual->control = (xreal *)malloc(sizeof(xreal) * f->
      length);
87
88        std::random_device rd;
89        std::mt19937 gen(rd());
90
91        xreal sigma = initialSigma;
92        if (i <= accurateSizeES)
93          sigma = sigma / denominator;
94        else if (i <= accurateSizeES + coarseSizeES)
95          sigma = sigma * denominator;
96
97        for (int j = 0; j < f->length; j++)
98          {
99            if (f->expression[j] == kInd)
100             {
101               std::normal_distribution<> d(f->parameters[j], sigma)
      ;
102               individual->parameters[j] = d(gen);
103               individual->control[j] = sigma;
104             }
105           else
106             {
107               individual->parameters[j] = -1;
108               individual->control[j] = -1;
109             }
110         }
111
112       pop->push_back(individual);
113     }
114
115   return pop;
116 }
```

Capítulo 4

Resultados

4.1. Entorno de pruebas

Las pruebas siguientes se han realizado sobre un ordenador personal con las siguientes características:

- Procesador[1]:
  - Modelo: Intel® Core™ i5-3330
  - Número de núcleos: 4
  - Velocidad de reloj: 3 GHz
  - Chache (Intel SmartCache): 6 MB
  - Direcciones de memoria: 64 bits
- Memoria RAM: 2 x 4GB, DDR3 Síncrona a 1600 MHz
- Sistema operativo GNU
  Linux Ubuntu 14.04 LTS, entorno XFCE, kernel 3.13.0-24-generic.
- Sistema operativo Microsoft Windows 7[2].

4.2. Series sin parametros

Se consideran inicialmente tres series simples, de las cuales se conocen expresiones sin parámetros.

- Sucesión de Fibonacci[3].

$$a(n) = \begin{cases} 0, & \text{si } n = 0 \\ 1, & \text{si } n = 1 \\ a(n-2) + a(n-1), & \text{si } n \geq 2 \end{cases}$$

[1] http://ark.intel.com/products/65509
[2] Empleado para las pruebas del software comercial Eureqa
[3] https://oeis.org/A000045

- Números triangulares[4].

  $a(n) = \frac{n(n-1)}{2}$

- Números poligonales centrales (Lazy Caterer en inglés)[5].

  $a(n) = a(n-1) + n$

En los tres casos, o bien no existe ningún parámetro, o es un parámetro "simple", que podemos componer con facilidad a partir del lenguaje propuesto (mediante las constantes 1 y 0, principalmente).

En el Listado 4.1 se puede ver el fragmento de nuestro fichero de tareas correspondientes a estas tres pruebas.

El conjunto de parámetros que utilizaremos para estas secuencias será el descrito en el Listado 4.2. No hay nada que destacar, este es un conjunto de parámetros obtenido por simple experimentación, y constituye el conjunto por defecto que empleamos en la mayoría de las pruebas.

Listado 4.1: Entrada para las tres series sin parámetros

```
1  name: A000045_fibonacci
2  dataset: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377,
       610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368,
       75025, 121393, 196418, 317811, 514229, 832040, 1346269,
       2178309, 3524578, 5702887, 9227465, 14930352, 24157817
3  settingsFile: param_set_default.txt
4
5  name: A000217_triangular
6  dataset: 0, 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105,
       120, 136, 153, 171, 190, 210, 231, 253, 276, 300, 325, 351,
       378, 406, 435, 465, 496, 528, 561, 595, 630, 666, 703, 741,
       780, 820, 861, 903, 946, 990, 1035, 1081, 1128, 1176, 1225,
       1275, 1326, 1378, 1431
7  settingsFile: param_set_default.txt
8
9  name: A000124_lazycaterer
10 dataset: 1, 2, 4, 7, 11, 16, 22, 29, 37, 46, 56, 67, 79, 92, 106,
       121, 137, 154, 172, 191, 211, 232, 254, 277, 301, 326, 352,
       379, 407, 436, 466, 497, 529, 562, 596, 631, 667, 704, 742,
       781, 821, 862, 904, 947, 991, 1036, 1082, 1129, 1177, 1226,
       1276, 1327, 1379
11 settingsFile: param_set_default.txt
```

Listado 4.2: param_set_default.txt

```
1  maxDepth, i, 3
2  probLeaf, f, 0.2
3  indInitVal, x, 1.0
4  maxPrev, i, 3
5  initialTargetLength, i, 1000
6  errorThreshold, s, 1.0
7  finishErrorThreshold, s, 0.005
8  errorFunction, e, kEFuncRel
9  errorMode, e, kErrorMean
10 evaluationMode, e, kEvalFullSeriesPredict
```

[4] https://oeis.org/A000217
[5] https://oeis.org/A000124

```
stepUpLag, i, 3
populationSize, i, 80
foreignersPercentage, f, 0.2
mutationPercentage, f, 0.3
crossoverPercentage, f, 0.3
elitistReplicationPercentage, f, 0.02
tournamentSize, i, 5
numIters, i, 5000
GPESItersRatio, i, 1
populationSizeES, i, 5
offspringES, i, 5
numItersES, i, 10
limitCyclesES, i, 20
coolDownCyclesES, i, 100
initialSigma, x, 2.0
minSigma, x, 0.00001
accurateSizeES, i, 1
coarseSizeES, i, 1
```

Los resultados obtenidos para las tres secuencias se encuentran en los listados Listado 4.3, Listado 4.4 y Listado 4.5. Podemos ver cómo en muy pocas iteraciones y tiempo se obtienen los resultados perfectos, con error 0, en el caso de los números poligonales centrales y los números triangulares, y error prácticamente nulo en el caso de Fibonacci.

Los resultados, ignorando los elementos neutros (divisiones por 1 y similares), son en los dos casos perfectos los esperados.

En el caso de Fibonacci, vemos que en menos de 2 segundos ha obtenido una aproximación perfectamente válida. Si consideramos los dos multiplicadores que ha ajustado para `P[1]` (el valor anterior de la serie), vemos que es equivalente, aproximadamente, a sumarlo sin más.

En este caso, la salida prematura se ha debido al margen de error que hemos exigido. En esta serie es factible alcanzar un valor de 0 perfecto, por lo que si modificamos el juego de parámetros especificando `finishErrorThreshold, s, 0.000000001`, podemos obligarle a iterar durante más tiempo, probablemente alcanzando la fórmula perfecta. Se puede ver el resultado en el Listado 4.6, donde en menos de 3 segundos se obtiene la susodicha fórmula.

Listado 4.3: Resultado para A000045_fibonacci

```
Final best individual:
Formula: K(-0.705908)P[1]*P[2]0++K(0.586199)P[1]/+#
(((P[1])/(K(0.586198543939)))+(((0)+(P[2]))+((P[1])*
(K(-0.705907656155)))))#
        ^                    ^     ^       ^       ^
                    ^
12333333333333333333333333322234444444443334444444
4444444444444444443210
Score: 7.45812e-06 evaluated over a length of 38 out of 38

Ran in 1.62983 seconds
for target data: A000045_fibonacci
```

Listado 4.4: Resultado para A000217_triangular

```
Final best individual:
```

```
Formula: T1/K(1)(-1)//P[1]-#
((P[1])-((((-1))/(K(1)))/((1)/(T))))#
        ^         ^       ^    ^       ^
12222222234444444444444433344444443210
Score: 0 evaluated over a length of 54 out of 54

Ran in 0.56233 seconds
for target data: A000217_triangular
```

Listado 4.5: Resultado para A000124_lazycaterer

```
Final best individual:
Formula: 00>K(1)T^1P[1]/+-#
((((P[1])/(1))+((T)^(K(1))))-((0)>(0)))#
          ^     ^     ^         ^     ^      ^
123444444444433344444444443222333333210
Score: 0 evaluated over a length of 53 out of 53

Ran in 0.577588 seconds
for target data: A000124_lazycaterer
```

Listado 4.6: Resultado para A000045_fibonacci_run2

```
Final best individual:
Formula: P[2]P[1]+#
((P[1])+(P[2]))#
        ^       ^
1222222222222210
Score: 0 evaluated over a length of 38 out of 38

Ran in 2.96872 seconds
for target data: A000045_fibonacci_run2
```

4.3. Series con parametros

Consideramos dos series con parámetros, ambas con grandes variaciones en el tamaño de las constantes.

- Números de Catalan[6]
  $a(n) = \frac{(2*n)!}{(n!*(n+1)!)}$
- Sucesión original BigParameters
  $a(n) = 1172888 * n + 0{,}982$

Se puede ver la entrada concreta en el Listado 4.7. La secuencia de Catalan emplea los mismos parámetros de la sección anterior (Listado 4.2) y se pueden ver los parámetros de control para la secuencia BigParameters en el Listado 4.8. En este caso, las diferencias son la eliminación de las variables referentes a los elementos anteriores de la serie (dada la corta longitud de la sucesión) y la alta exigencia al error final cometido.

[6] https://oeis.org/A000108

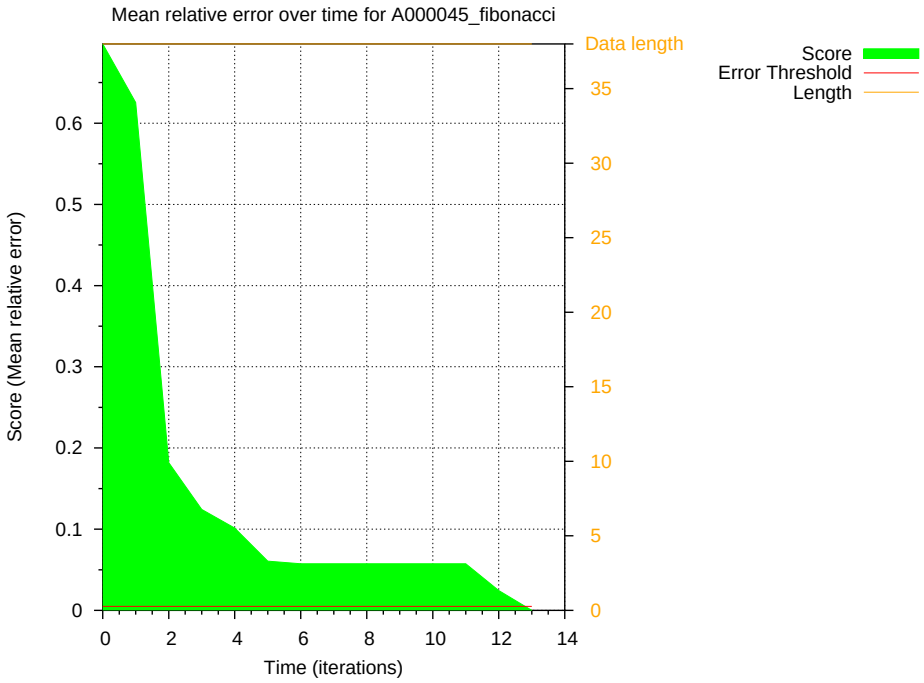

Figura 4.1: Evolución de A000045_fibonacci

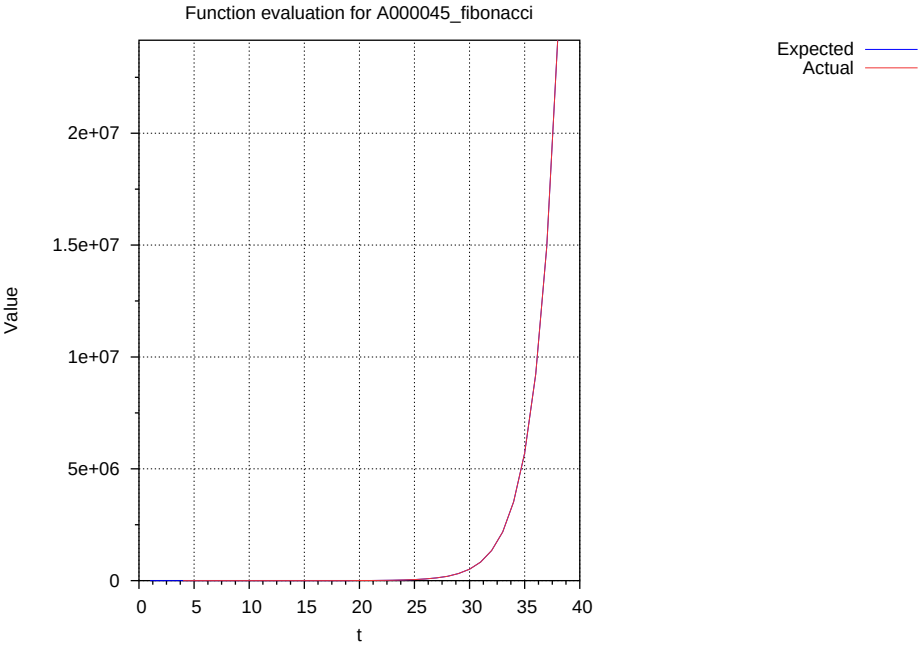

Figura 4.2: Resultado de A000045_fibonacci

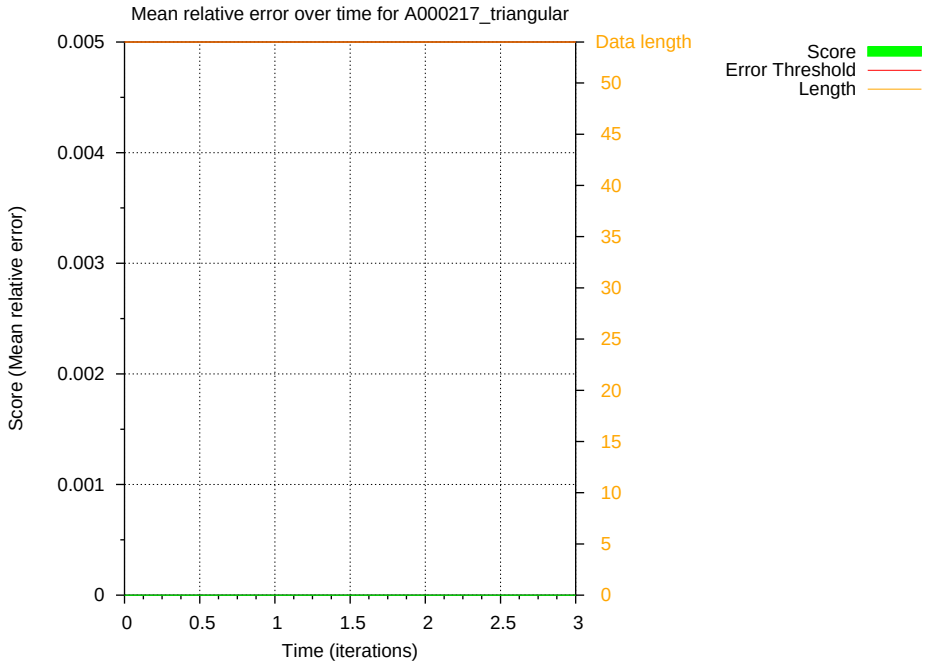

Figura 4.3: Evolución de A000217_triangular

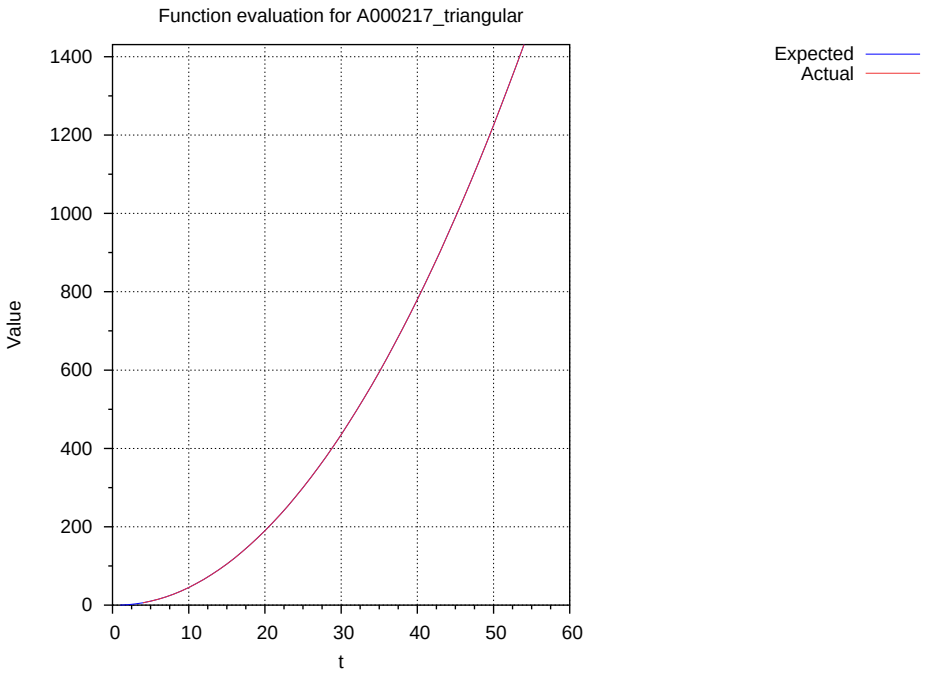

Figura 4.4: Resultado de A000217_triangular

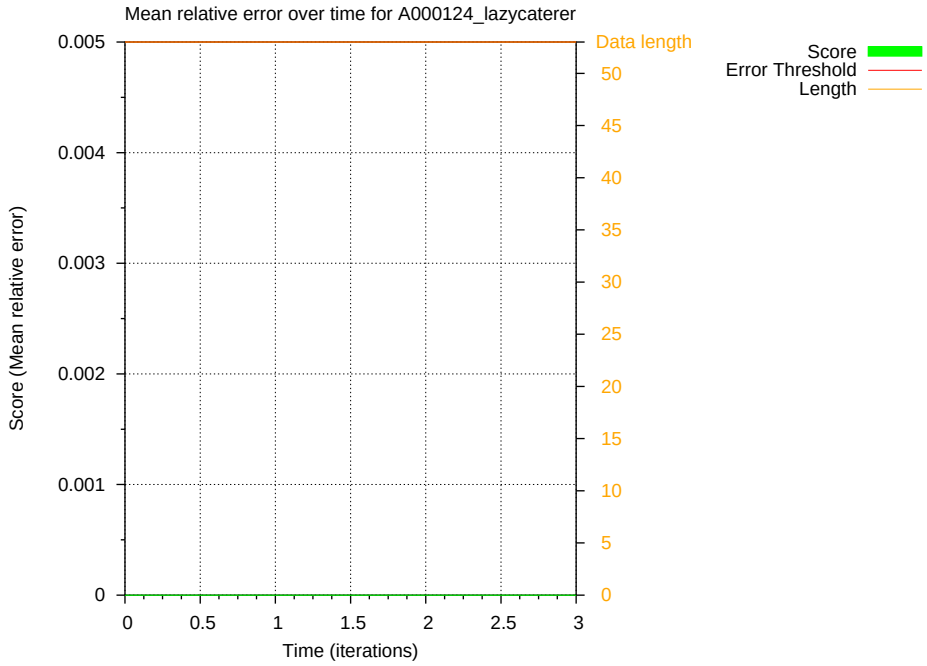

Figura 4.5: Evolución de A000124_lazycaterer

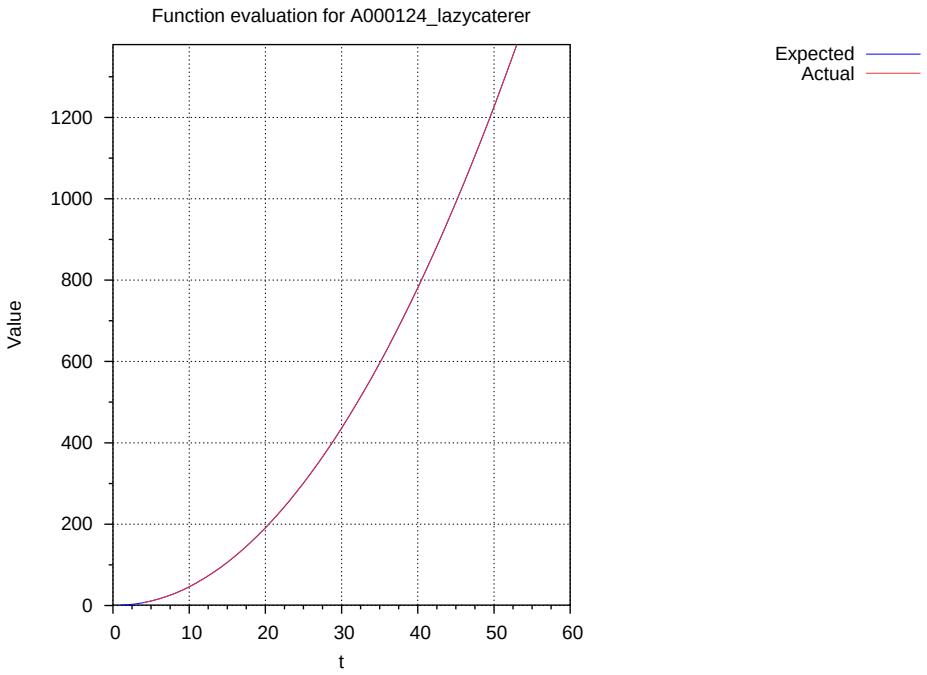

Figura 4.6: Resultado de A000124_lazycaterer

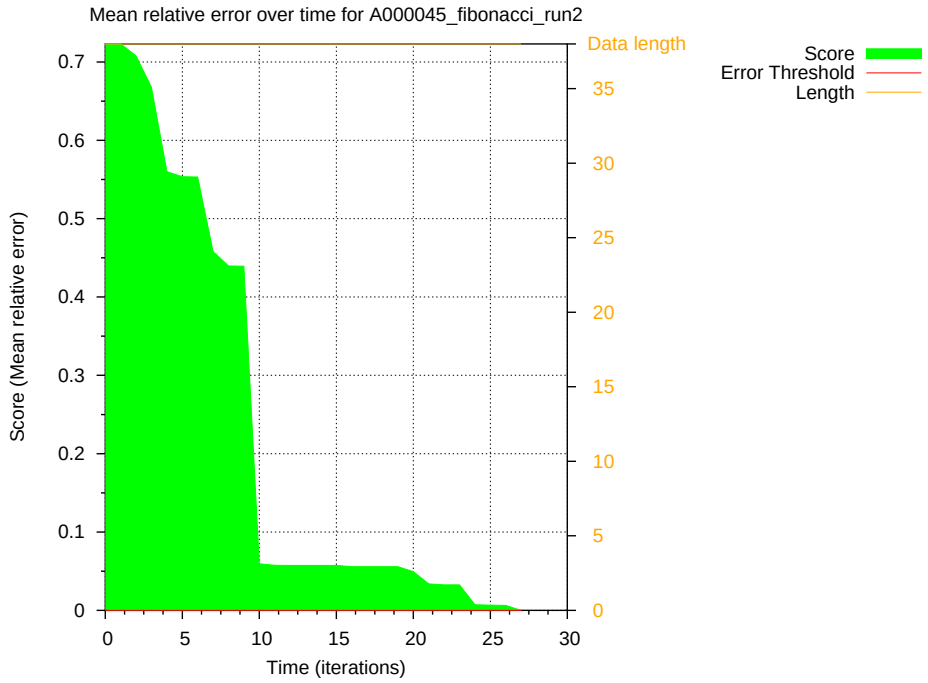

Figura 4.7: Evolución de A000045_fibonacci_run2

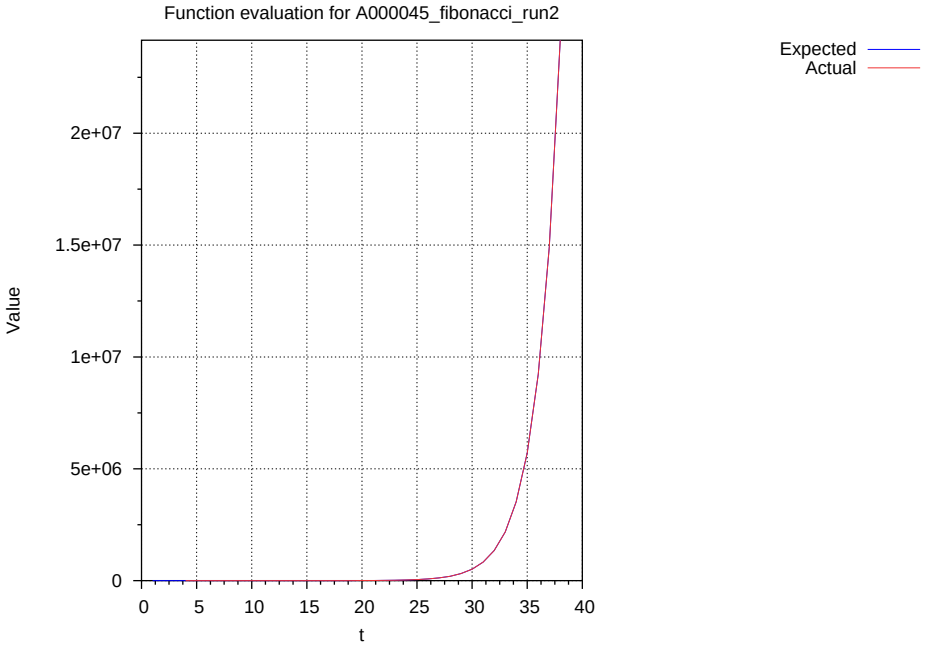

Figura 4.8: Resultado de A000045_fibonacci_run2

Listado 4.7: Entrada para las dos series con parámetros

```
name: A000108_catalan
dataset: 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786,
    208012, 742900, 2674440, 9694845, 35357670, 129644790,
    477638700, 1767263190, 6564120420, 24466267020, 91482563640,
    343059613650, 1289904147324, 4861946401452, 18367353072152,
    69533550916004, 263747951750360, 1002242216651368,
    3814986502092304
settingsFile: param_set_default.txt

name: X000003_bigParameters
dataset: 0.982,1172888.982,2345776.982,3518664.982,4691552.982
settingsFile: param_set_bigParameters.txt
```

Listado 4.8: param_set_bigParameters.txt

```
maxDepth, i, 3
probLeaf, f, 0.3
indInitVal, x, 1.0
maxPrev, i, 0
initialTargetLength, i, 1000
errorThreshold, s, 1.0
finishErrorThreshold, s, 0.0000001
errorFunction, e, kEFuncRel
errorMode, e, kErrorMean
evaluationMode, e, kEvalFullSeriesPredict
stepUpLag, i, 3
populationSize, i, 80
foreignersPercentage, f, 0.2
mutationPercentage, f, 0.3
crossoverPercentage, f, 0.3
elitistReplicationPercentage, f, 0.02
tournamentSize, i, 5
numIters, i, 5000
GPESItersRatio, i, 1
populationSizeES, i, 5
offspringES, i, 5
numItersES, i, 10
limitCyclesES, i, 20
coolDownCyclesES, i, 100
initialSigma, x, 2.0
minSigma, x, 0.00001
accurateSizeES, i, 1
coarseSizeES, i, 1
```

El resultado para la sucesión BigParameters (Listado 4.10) es casi instantáneo y da una buena idea de la capacidad de optimización de la estrategia evolutiva implementada. Como se puede ver, los dos parámetros se han ajusto con una precisión notable (6 cifras significativas en ambos casos), pese a las diferentes magnitudes de ambos.

Para Catalan, el resultado (Listado 4.9) es bastante peor, como era de esperar. Al no tener implementado el lenguaje un operador sumatorio, factorial o similares que permitiesen arrojar la fórmula en cuestión, el algoritmo ha realizado una aproximación razonable. Además, el mayor elemento de la secuencia tiene 14 cifras decimales, lo cual supone una exigencia de precisión muy elevada para las constantes involucradas en la fórmula.

El resultado final propuesto por el algoritmo llega a un 1.2 % de error medio pese a esto, por lo que dada las circunstancias puede considerarse un buen resultado. En este caso, eso sí, se obtiene agotando la totalidad de las 5000 iteraciones y rozando lo 10 minutos de tiempo de computación. La mayor parte del cómputo se realiza en las primeras 500 iteraciones (la décima parte del total), por lo que de implementar una guarda de parada menos exigente podríamos haber acabado bastante más rápido.

Listado 4.9: Resultado para A000108_catalan

```
Final best individual:
Formula: TP[2]*K(0.784942)-TK(-4.43181)-K(1.07227)K(17.
9881)//*#
((((K(17.9880904253))/(K(1.07227460415)))/((K(-4.431812
57894))-(T)))*((K(0.784942016247))-((P[2])*(T))))#
                    ^                   ^
       ^     ^                    ^       ^      ^
12344444444444444444444444444444444444433344444444444
4444444444432223333333333333333333344444444443210
Score: 0.0121637 evaluated over a length of 31 out of 31

Ran in 520.921 seconds
for target data: A000108_catalan
```

Listado 4.10: Resultado para X000003_bigParameters

```
Final best individual:
Formula: K(0.982)TK(1.17289e+06)*+#
(((K(1172888.02655))*(T))+(K(0.981999782091)))#
                    ^     ^                   ^
12333333333333333333332222222222222222222210
Score: 6.24917e-08 evaluated over a length of 5 out of 5

Ran in 0.222001 seconds
for target data: X000003_bigParameters
```

4.4. Sucesión de Recamán

Consideramos ahora la sucesión de Recamán[7]. En este caso, la secuencia no responde a una fórmula simple, si no que se construye algorítmicamente. En concreto:

- $a(0) = 0$
- Para cualquier número positivo por encima de 0, $a(n) = a(n-1) - n$ si el número es positivo y no está aún en la serie. En otro caso, $a(n) = a(n-1) + n$.

El chequeo de si un número está o no en la serie hace muy compleja la regresión en un lenguaje aritmético como el empleado en esta memoria.

[7] https://oeis.org/A005132

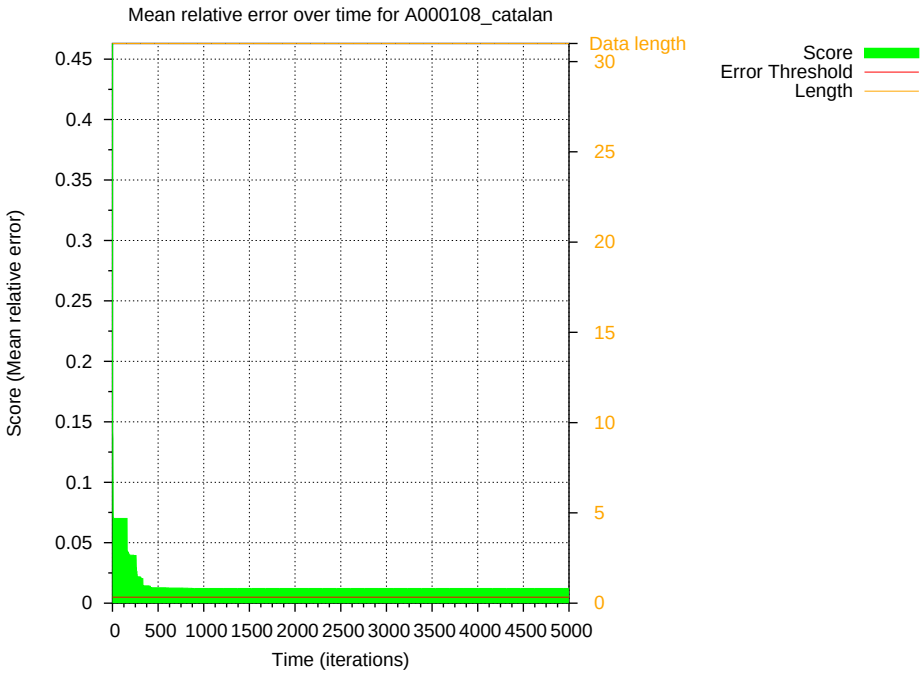

Figura 4.9: Evolución de A000108_catalan

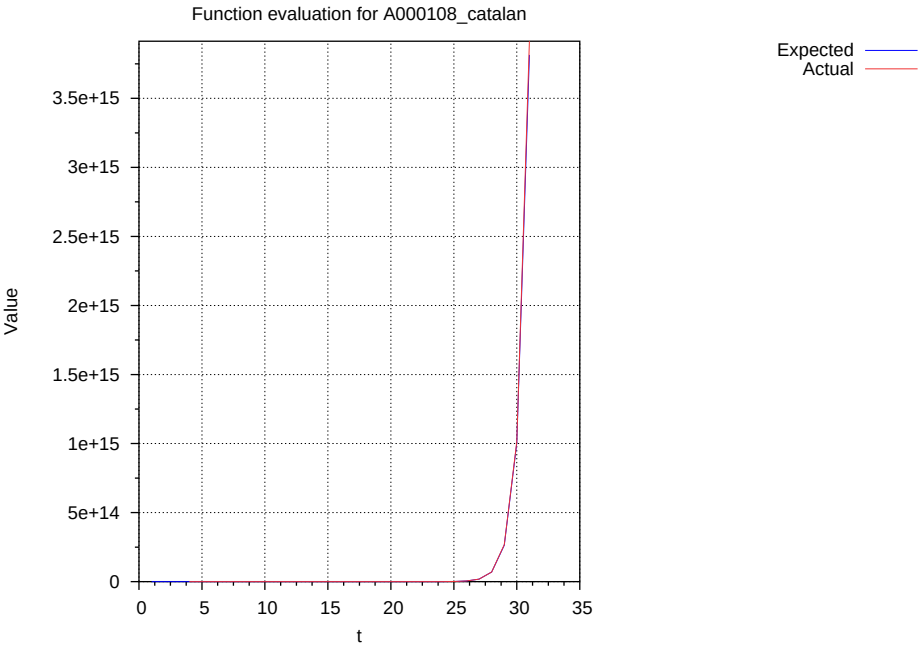

Figura 4.10: Resultado de A000108_catalan

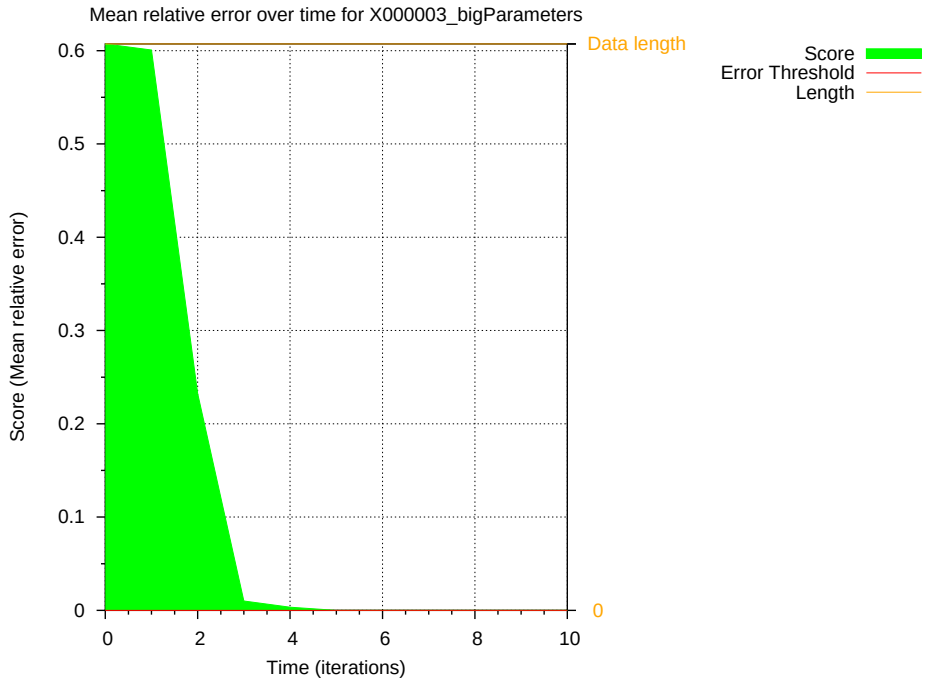

Figura 4.11: Evolución de X000003_bigParameters

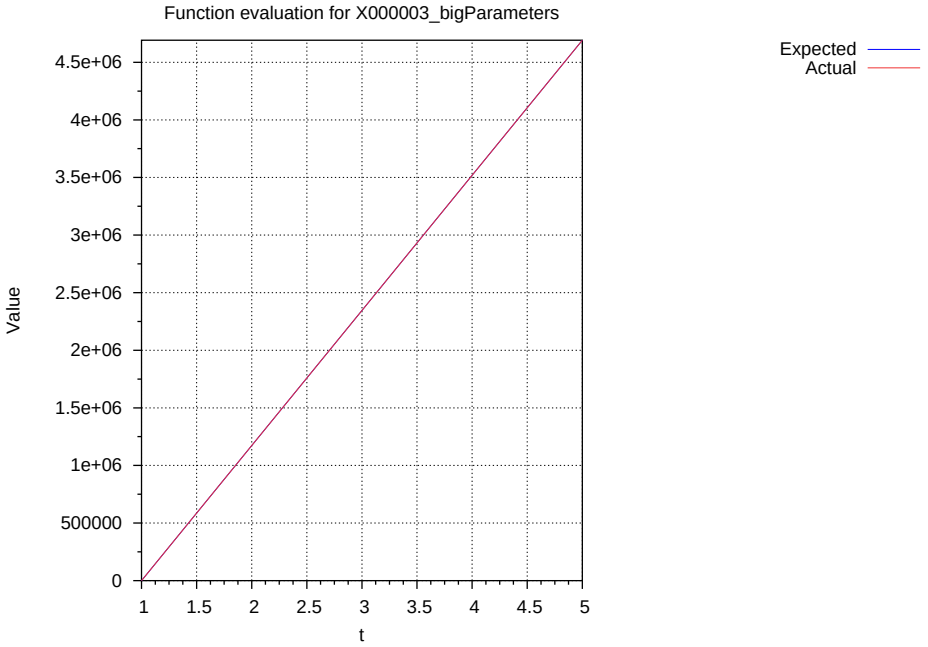

Figura 4.12: Resultado de X000003_bigParameters

Se puede ver la entrada empleada en el Listado 4.11. Primero hemos probado el resultado empleando los parámetros por defecto ya presentados (Listado 4.2) y después hemos ejecutado una segunda prueba con unos parámetros modificados para la secuencia (Listado 4.12). Los parámetros específicos son iguales a los por defecto, con la única diferencia de que empleamos el método de evaluación `kEvalSinglePredict`, en vez de `kEvalFullSeriesPredict`. La diferencia entre ambas se detalla en la Subsección 3.6.6: Evaluation.

Listado 4.11: Entrada para la sucesión de Recamán

```
1 name: A005132_recaman
2 dataset: 0, 1, 3, 6, 2, 7, 13, 20, 12, 21, 11, 22, 10, 23, 9, 24,
      8, 25, 43, 62, 42, 63, 41, 18, 42, 17, 43, 16, 44, 15, 45, 14,
      46, 79, 113, 78, 114, 77, 39, 78, 38, 79, 37, 80, 36, 81, 35,
      82, 34, 83, 33, 84, 32, 85, 31, 86, 30, 87, 29, 88, 28, 89, 27,
       90, 26, 91, 157, 224, 156, 225, 155
3 settingsFile: param_set_default.txt
4
5 name: A005132_recaman_singlePredict
6 dataset: 0, 1, 3, 6, 2, 7, 13, 20, 12, 21, 11, 22, 10, 23, 9, 24,
      8, 25, 43, 62, 42, 63, 41, 18, 42, 17, 43, 16, 44, 15, 45, 14,
      46, 79, 113, 78, 114, 77, 39, 78, 38, 79, 37, 80, 36, 81, 35,
      82, 34, 83, 33, 84, 32, 85, 31, 86, 30, 87, 29, 88, 28, 89, 27,
       90, 26, 91, 157, 224, 156, 225, 155
7 settingsFile: param_set_singlePredict.txt
```

Listado 4.12: param_set_singlePredict.txt

```
1  maxDepth, i, 3
2  probLeaf, f, 0.2
3  indInitVal, x, 1.0
4  maxPrev, i, 3
5  initialTargetLength, i, 1000
6  errorThreshold, s, 1.0
7  finishErrorThreshold, s, 0.005
8  errorFunction, e, kEFuncRel
9  errorMode, e, kErrorMean
10 evaluationMode, e, kEvalSinglePredict
11 stepUpLag, i, 3
12 populationSize, i, 80
13 foreignersPercentage, f, 0.2
14 mutationPercentage, f, 0.3
15 crossoverPercentage, f, 0.3
16 elitistReplicationPercentage, f, 0.02
17 tournamentSize, i, 5
18 numIters, i, 5000
19 GPESItersRatio, i, 1
20 populationSizeES, i, 5
21 offspringES, i, 5
22 numItersES, i, 10
23 limitCyclesES, i, 20
24 coolDownCyclesES, i, 100
25 initialSigma, x, 2.0
26 minSigma, x, 0.00001
27 accurateSizeES, i, 1
28 coarseSizeES, i, 1
```

El resultado para los parámetros por defecto (Listado 4.13) está bastante lejos de la función original, con un error relativo medio sobre el 30 % de los valores

originales. Si estudiamos la gráfica (Figura 4.14), vemos que la función llega a ajustar algunas características de la original, pero dista aún así de ser una gran aproximación.

Con el segundo juego de parámetros (Listado 4.14), la cosa cambia. El resultado, pese a tener un porcentaje de error medio del 12 %, es bastante acertado si vemos la gráfica (Figura 4.16). Es importante recordar que este resultado no permite **reconstruir** la gráfica, sólo predecir el siguiente valor cuando se conocen ya los anteriores. El error que se indica corresponde a este tipo de predicción a partir de la secuencia original, y no refleja un error real a la hora de predecir la serie completa. Sin embargo, sí que nos permite ver un patrón, una tendencia en la sucesión original. La mayor parte de los elementos cumplen ésta función con total precisión, si bien existen puntos donde el resultado es bastante distinto.

Listado 4.13: Resultado para A005132_recaman

```
Formula: K(32.5488)T-K(129.97)P[2]%/TK(-1.05949)^T-+#
(((T)-((K(-1.05948709046))^(T)))+(((P[2])%(K(129.9696
36902)))/((T)-(K(32.5487674566)))))#
     ^                    ^     ^          ^
        ^    ^                      ^
12333334444444444444444444444432223444444444444444444
444444333444444444444444444444443210
Score: 0.308641 evaluated over a length of 71 out of 71

Ran in 953.256 seconds
for target data: A005132_recaman
```

Listado 4.14: Resultado para A005132_recaman_singlePredict

```
Final best individual:
Formula: P[2]K(63.9549)-K(7.81546)logP[1]P[2]/P[2]K(-
1.00116)%++#
((((K(-1.00116234479))%(P[2]))+((P[2])/(P[1])))+((K(7
.81546026921))log((K(63.9548828295))-(P[2]))))#
                  ^        ^        ^         ^
               ^^^                  ^         ^
12344444444444444444444444444333444444444444432223333
33333333333333333444444444444444444444443210
Score: 0.129795 evaluated over a length of 71 out of 71

Ran in 905.249 seconds
for target data: A005132_recaman_singlePredict
```

4.5. Sucesión segmentada

Consideramos ahora una función construida a tramos, de forma segmentada:

- $a(n) = 0$ para $n < 20$
- $a(n) = a(n-1) + 1$ para $20 \leq n < 35$
- $a(n) = 0$ para $35 \leq n$

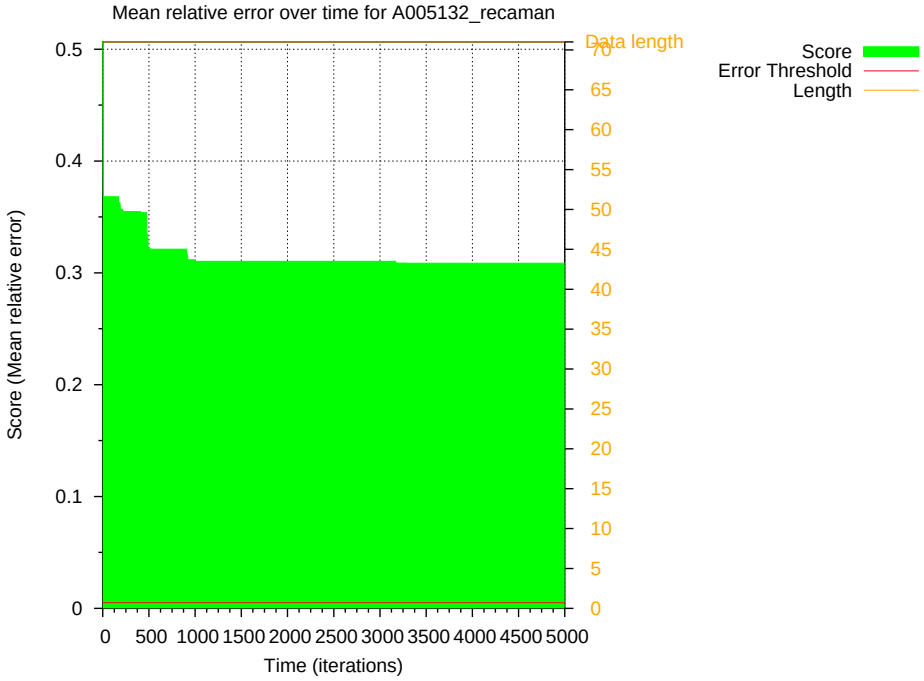

Figura 4.13: Evolución de A005132_recaman

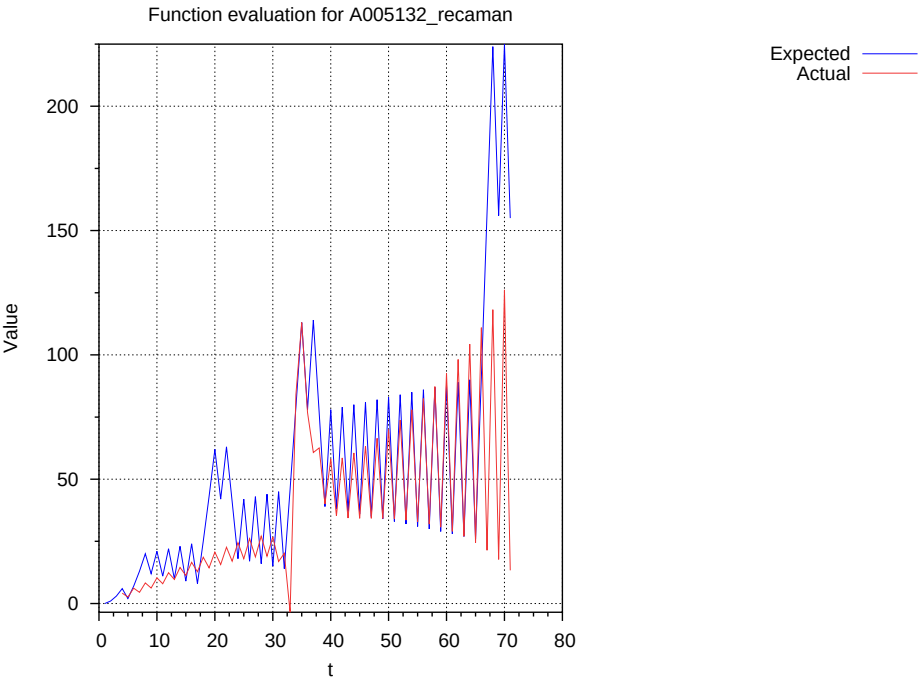

Figura 4.14: Resultado de A005132_recaman

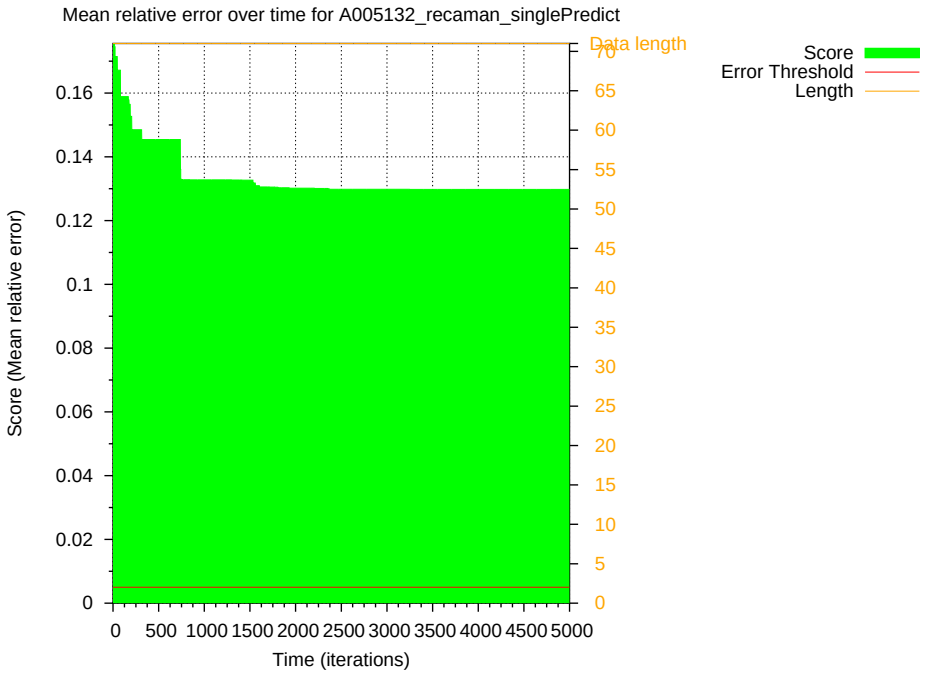

Figura 4.15: Evolución de A005132_recaman_singlePredict

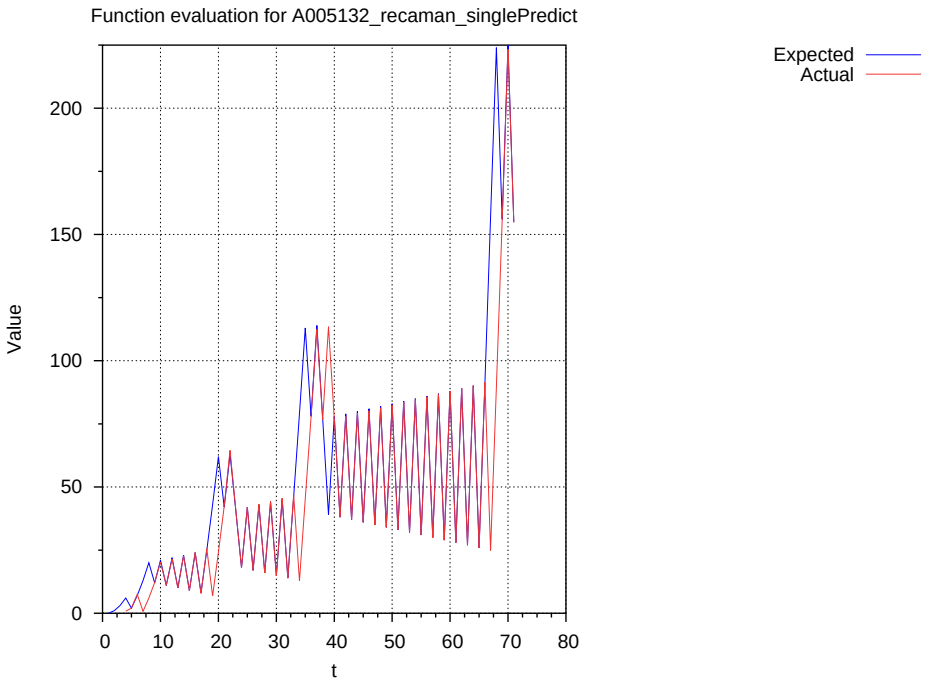

Figura 4.16: Resultado de A005132_recaman_singlePredict

En este caso, se trata de la composición de 3 funciones simples, dos constantes y una lineal creciente.

Se puede ver la entrada empleada en el Listado 4.15. De nuevo, se han empleado los parámetros por defecto (Listado 4.2).

Listado 4.15: Entrada para la sucesión segmentada

```
name: X000001_segmented
dataset: 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,3,
4,5,6,7,8,9,10,11,12,13,14,15,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0
settingsFile: param_set_default.txt
```

El resultado para los parámetros por defecto (Listado 4.16), como se puede observar, es prácticamente perfecto. La fórmula separa claramente los fragmentos (la primera parte del producto resulta 0 si estamos en el primer o tercer tramo, y en otro caso actúa la segunda parte).

Como podemos apreciar, en la práctica el empleo de la comparación, al retornar 0 o 1, es equivalente a disponer de instrucciones condicionales, lo que permite ajustar funciones bastante más complejas que mediante regresión clásica.

Listado 4.16: Resultado para X000001_segmented

```
Formula: TK(0.233583)/K(19.6722)T%-K(39.7939)K(58.8252)
-K(35.9295)T%>*#
((((T)%(K(35.9295204867)))>((K(58.8252278424))-(K(39.79
39351891))))*(((T)%(K(19.6722026324)))-((K(0.2335831770
38))/(T))))#
      ^                   ^                     ^
            ^     ^                   ^
    ^       ^
1234444444444444444444444333444444444444444444444444444
4444444443222344444444444444444444443334444444444444444
444444443210
Score: 0.000649626 evaluated over a length of 52 out of 52

Ran in 48.7245 seconds
for target data: X000001_segmented
```

4.6. Sucesión alternada

Consideramos ahora una sucesión alternada, entrelazando tres sucesiones distintas:

- $a(n) = 0$ para $n \equiv 0 \mathrm{mod} 3$
- $a(n) = 2$ para $n \equiv 1 \mathrm{mod} 3$
- $a(2) = 0$
- $a(n) = a(n - 3$ para $n \equiv 2 \mathrm{mod} 3, n > 2$

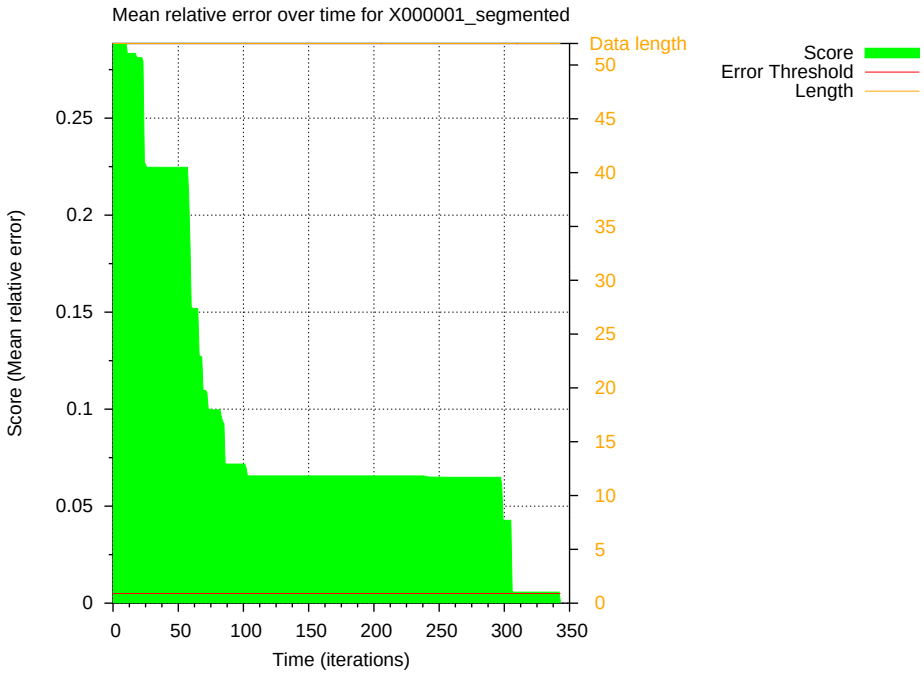

Figura 4.17: Evolución de X000001_segmented

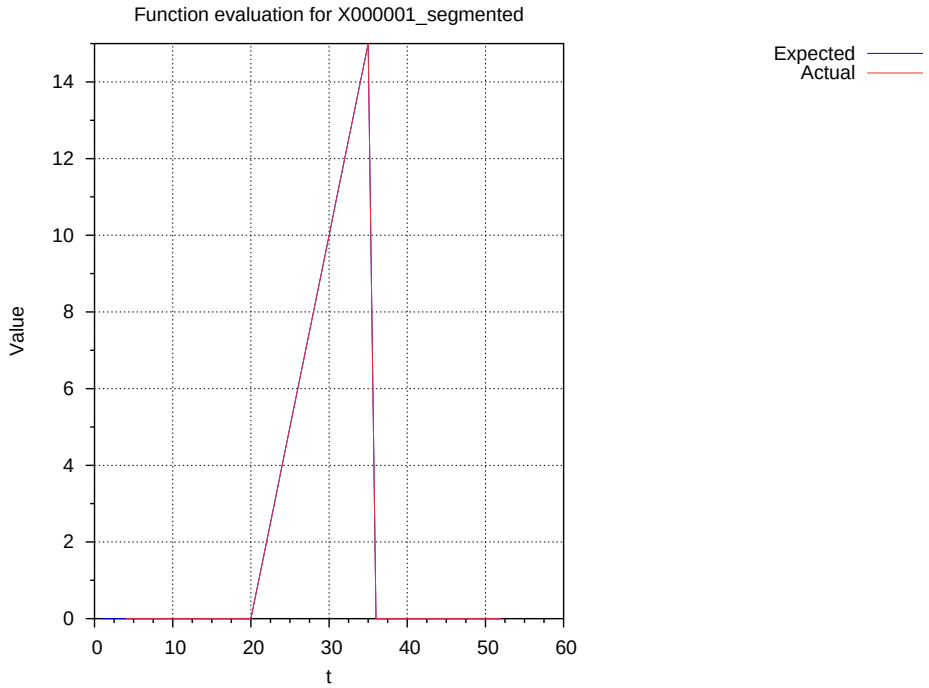

Figura 4.18: Resultado de X000001_segmented

De nuevo, hemos empleado la composición de 3 funciones simples, dos constantes y una lineal creciente, alternándolas entre sí en cada paso de la secuencia.

Se puede ver la entrada empleada en el Listado 4.17. De nuevo, se han empleado los parámetros por defecto (Listado 4.2).

Listado 4.17: Entrada para la sucesión alternada

```
name: X000002_alternated
dataset: 0,4,1,0,4,2,0,4,3,0,4,4,0,4,5,0,4,6,0,4,7,0,4,
8,0,4,9,0,4,10,0,4,11,0,4,12,0,4,13,0,4,14,0,4,15,0,4,
16,0,4,17,0,4,18,0,4,19,0,4,20,0,4,21,0,4
settingsFile: param_set_default.txt
```

El resultado para los parámetros por defecto (Listado 4.16), es perfecto. En este caso, no sólo obtiene error 0, sino que encuentra una fórmula sorprendentemente simple para definir la función propuesta.

Listado 4.18: Resultado para X000002_alternated

```
Final best individual:
Formula: 0P[2]/P[3]+#
((P[3])+((P[2])/(0)))#
        ^       ^     ^
12222222233333333210
Score: 0 evaluated over a length of 65 out of 65

Ran in 0.486321 seconds
for target data: X000002_alternated
```

4.7. Números primos

Consideramos ahora la sucesión matemática por excelencia, la sucesión de los números primos[8]. No tenemos ningún tipo de fórmula para la construcción de la secuencia, dependiendo exclusivamente de métodos algorítmicos más o menos exhaustivos para generarla.

Se puede ver la entrada empleada en el Listado 4.19. Hemos realizado dos pruebas, la primera empleando los parámetros por defecto ya presentados (Listado 4.2) y la segunda empleando unos parámetros distintos (Listado 4.20). Este segundo juego de parámetros difiere del habitual en el empleo de errores absolutos en vez de relativos. En este caso, como la secuencia no crece de forma exagerada, no corremos el riesgo de que los errores de los mayores elementos oculten los de los pequeños..

Listado 4.19: Entrada para la sucesión de los números primos

```
name: A000040_prime
dataset: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47,
    53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109,
    113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179,
    181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241,
    251, 257, 263, 269, 271
```

[8] https://oeis.org/A000040

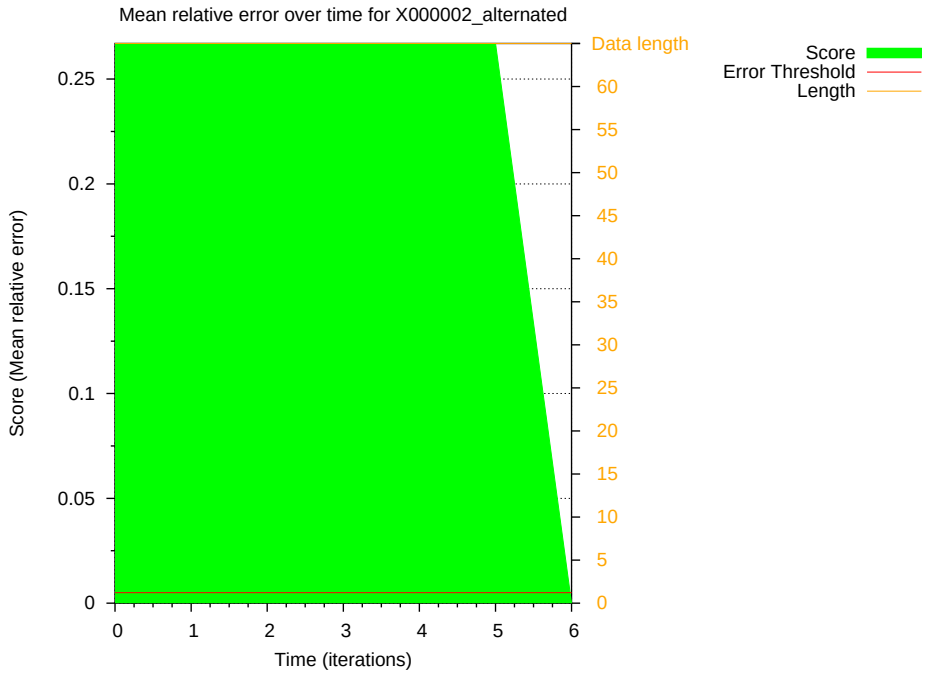

Figura 4.19: Evolución de X000002_alternated

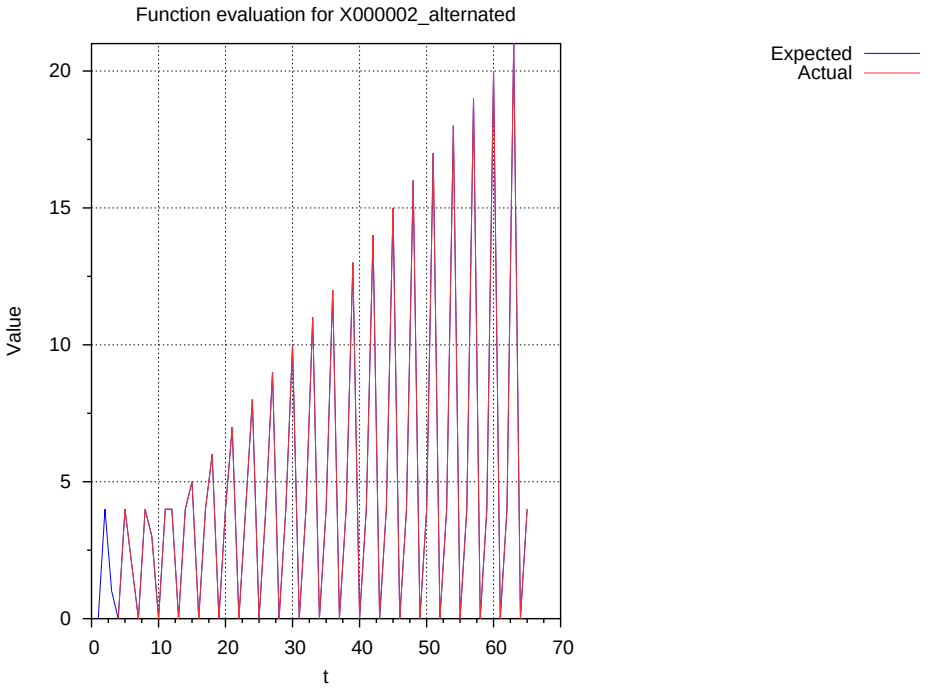

Figura 4.20: Resultado de X000002_alternated

```
3 settingsFile: param_set_default.txt
4 end
5
6 name: A000040_prime_absoluteError
7 dataset: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47,
      53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109,
      113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179,
      181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241,
      251, 257, 263, 269, 271
8 settingsFile: param_set_absoluteError.txt
```

Listado 4.20: param_set_absoluteError.txt

```
1  maxDepth, i, 3
2  probLeaf, f, 0.2
3  indInitVal, x, 1.0
4  maxPrev, i, 3
5  initialTargetLength, i, 1000
6  errorThreshold, s, 1.0
7  finishErrorThreshold, s, 0.005
8  errorFunction, e, kEFuncAbs
9  errorMode, e, kErrorMax
10 evaluationMode, e, kEvalFullSeriesPredict
11 stepUpLag, i, 3
12 populationSize, i, 80
13 foreignersPercentage, f, 0.2
14 mutationPercentage, f, 0.3
15 crossoverPercentage, f, 0.3
16 elitistReplicationPercentage, f, 0.02
17 tournamentSize, i, 5
18 numIters, i, 5000
19 GPESItersRatio, i, 1
20 populationSizeES, i, 5
21 offspringES, i, 5
22 numItersES, i, 10
23 limitCyclesES, i, 20
24 coolDownCyclesES, i, 100
25 initialSigma, x, 2.0
26 minSigma, x, 0.00001
27 accurateSizeES, i, 1
28 coarseSizeES, i, 1
```

El resultado para los parámetros por defecto (Listado 4.21), empleando el error relativo como medida de error, como podemos observar es bastante ajustado, pero algo "plano". En total cometemos apenas un 1.3 % de error relativo, pero observando la gráfica (Figura 4.22) podemos ver que el resultado proviene de una regresión prácticamente plana, que omite todo el interés de la serie. El resultado sería formidable para una serie con error, pero en nuestro caso resulta bastante menos interesante.

Con el segundo juego de parámetros (Listado 4.22), empleando el error absoluto como medida del error, en cambio, nos acercamos mucho más a la secuencia real de los números primos. El error medio cometido es de 4, bastante elevado, pero en la gráfica (Figura 4.24) podemos observar que se amolda mejor a la sucesión, separándose de una regresión lineal simple.

Listado 4.21: Resultado para A000040_prime

```
Final best individual:
Formula: K(1.331)T^TK(-0.311576)++K(0.521536)T^TP[1]+%+#
((((P[1])+(T))%((T)^(K(0.521536032318))))+(((K(-0.31157
6224497))+(T))+((T)^(K(1.33100327144)))))#
          ^     ^     ^                     ^
          ^     ^     ^                     ^
12344444444443334444444444444444444444432223444444444444
44444444444443334444444444444444444444443210
Score: 0.012927 evaluated over a length of 58 out of 58

Ran in 803.012 seconds
for target data: A000040_prime
```

Listado 4.22: Resultado para A000040_prime_absoluteError

```
Final best individual:
Formula: K(1.33033)T^TK(0.890473)--(-1)K(7.33697)+K(1.3
3869)T^%-#
((((T)^(K(1.33868747454)))%((K(7.33696914778))+((-1))))
-(((K(0.890473057725))-(T))-((T)^(K(1.33032733716)))))#
      ^                   ^                     ^
^                   ^     ^     ^                 ^
12344444444444444444444433344444444444444444444444432
22344444444444444444444433344444444444444444444443210
Score: 4.34971 evaluated over a length of 58 out of 58

Ran in 835.996 seconds
for target data: A000040_prime_absoluteError
```

4.8. Números de Gödel

Consideramos por último una sucesión matemática compuesta por la función de Gödel[8] de la posición en la serie del elemento[9].

Esta sucesión permite localizar números de Meertens[1], para los que se cumple que el número original es igual a su codificación de Gödel, donde la codificación de Gödel se define como $a(n) = 2^d(1)*3^d(2)*...*prime(k)^d(k)$, donde $d(1)d(2)...d(k)$ es la representación decimal del número original n. Construyendo esta secuencia, cualquier elemento cuyo valor sea igual a su índice será un número de Meertens.

Se puede ver la entrada que hemos empleado en el Listado 4.23. Hemos realizado dos pruebas, con dos juegos de parámetros distintos. La primera emplea el conjunto de parámetros del **??** y la segunda el del **??**. En ambos casos empleamos el máximo error absoluto como medida del error para la secuencia, siendo la principal diferencia la longitud de entrada considerada. En el primer juego de parámetros consideramos desde el principio la totalidad de la secuencia, mientras que en el segundo empezamos desde una versión reducida y vamos aumentando la longitud progresivamente.

Listado 4.23: Entrada para los números de Gödel

```
name: A189398_meertens_fullseries
```

[9] https://oeis.org/A189398

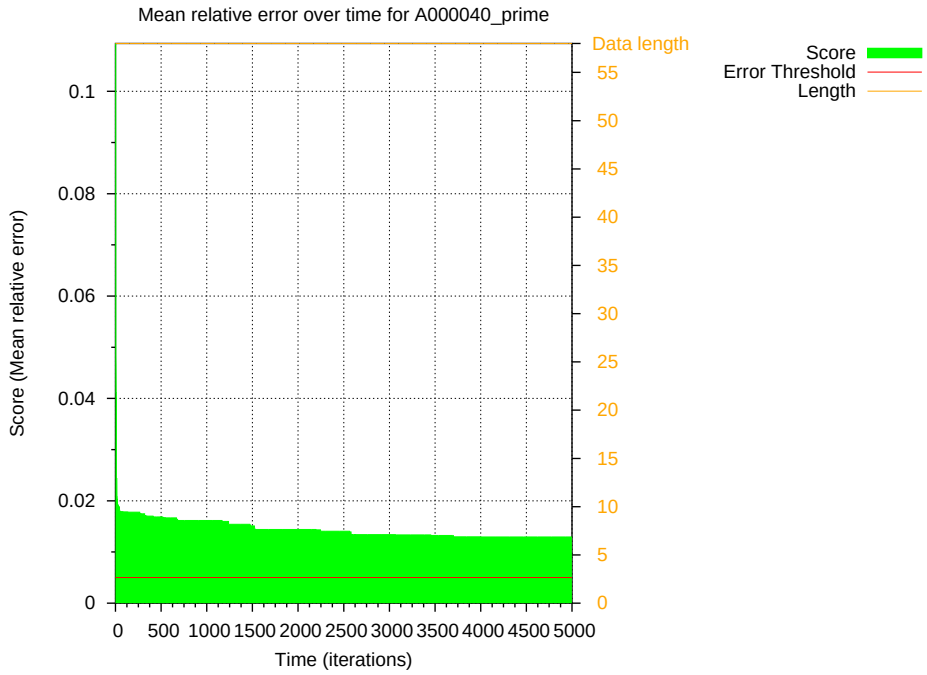

Figura 4.21: Evolución de A000040_prime

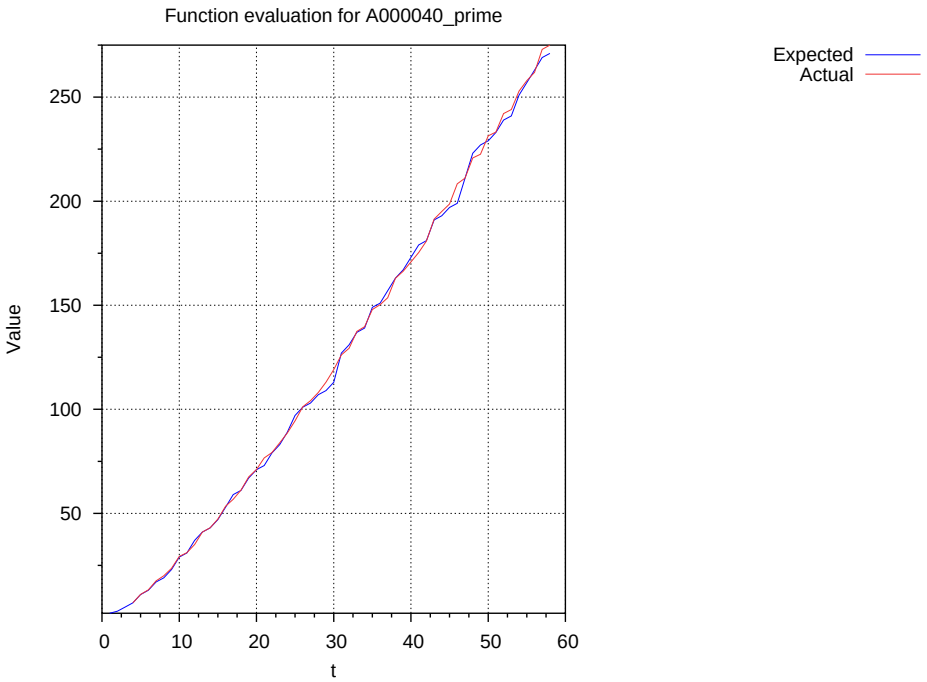

Figura 4.22: Resultado de A000040_prime

Figura 4.23: Evolución de A000040_prime_absoluteError

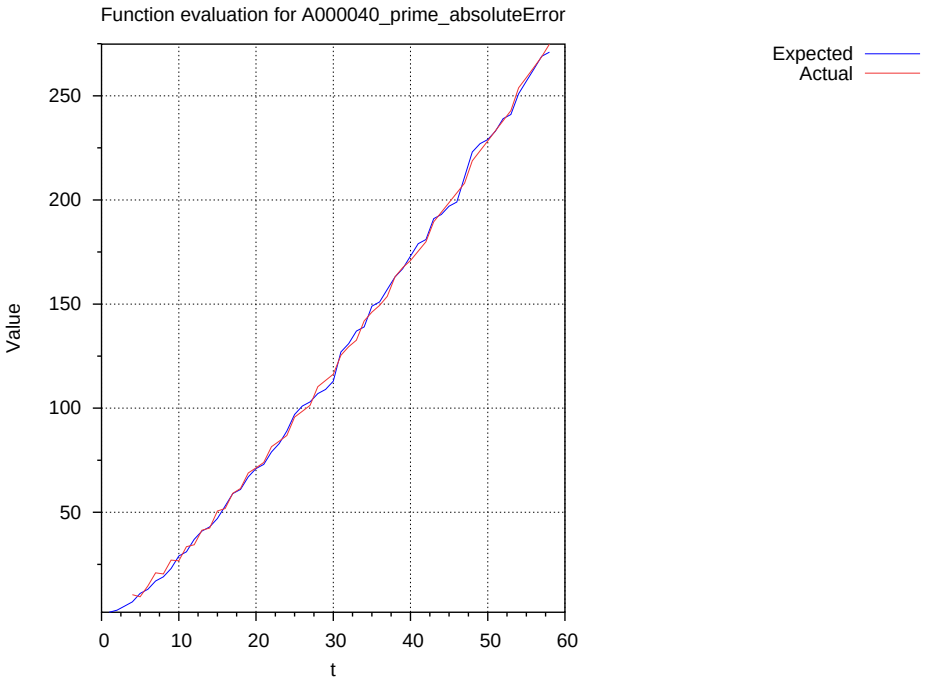

Figura 4.24: Resultado de A000040_prime_absoluteError

```
2 dataset:        2, 4, 8, 16, 32, 64, 128, 256, 512, 2, 6, 18, 54,
      162, 486, 1458, 4374, 13122, 39366, 4, 12, 36, 108, 324, 972,
      2916, 8748, 26244, 78732, 8, 24, 72, 216, 648, 1944, 5832,
      17496, 52488, 157464, 16, 48, 144, 432, 1296, 3888, 11664,
      34992, 104976, 314928, 32
3 settingsFile: param_set_merteen.txt
4
5 name: A189398_meertens_stepup
6 dataset:        2, 4, 8, 16, 32, 64, 128, 256, 512, 2, 6, 18, 54,
      162, 486, 1458, 4374, 13122, 39366, 4, 12, 36, 108, 324, 972,
      2916, 8748, 26244, 78732, 8, 24, 72, 216, 648, 1944, 5832,
      17496, 52488, 157464, 16, 48, 144, 432, 1296, 3888, 11664,
      34992, 104976, 314928, 32
7 settingsFile: param_set_merteen_stepup.txt
```

Listado 4.24: param_set_merteen.txt

```
1  maxDepth, i, 4
2  probLeaf, f, 0.2
3  indInitVal, x, 1.0
4  maxPrev, i, 4
5  initialTargetLength, i, 1000
6  errorThreshold, s, 0.0
7  finishErrorThreshold, s, 0.005
8  errorFunction, e, kEFuncAbs
9  errorMode, e, kErrorMax
10 evaluationMode, e, kEvalFullSeriesPredict
11 stepUpLag, i, 5
12 populationSize, i, 80
13 foreignersPercentage, f, 0.2
14 mutationPercentage, f, 0.3
15 crossoverPercentage, f, 0.3
16 elitistReplicationPercentage, f, 0.02
17 tournamentSize, i, 5
18 numIters, i, 5000
19 GPESItersRatio, i, 1
20 populationSizeES, i, 8
21 offspringES, i, 8
22 numItersES, i, 10
23 limitCyclesES, i, 20
24 coolDownCyclesES, i, 100
25 initialSigma, x, 2.0
26 minSigma, x, 0.00001
27 accurateSizeES, i, 1
28 coarseSizeES, i, 1
```

Listado 4.25: param_set_merteen_stepup.txt

```
1  maxDepth, i, 4
2  probLeaf, f, 0.2
3  indInitVal, x, 1.0
4  maxPrev, i, 4
5  initialTargetLength, i, 15
6  errorThreshold, s, 8000000
7  finishErrorThreshold, s, 0.005
8  errorFunction, e, kEFuncAbs
9  errorMode, e, kErrorMax
10 evaluationMode, e, kEvalFullSeriesPredict
11 stepUpLag, i, 10
12 populationSize, i, 80
13 foreignersPercentage, f, 0.2
```

```
mutationPercentage, f, 0.3
crossoverPercentage, f, 0.3
elitistReplicationPercentage, f, 0.02
tournamentSize, i, 5
numIters, i, 5000
GPESItersRatio, i, 1
populationSizeES, i, 8
offspringES, i, 8
numItersES, i, 10
limitCyclesES, i, 20
coolDownCyclesES, i, 100
initialSigma, x, 2.0
minSigma, x, 0.00001
accurateSizeES, i, 1
coarseSizeES, i, 1
```

Los resultados no son particularmente notables en ninguno de los dos casos, teniendo un error absoluto máximo bastante elevado. Sin embargo, es interesante ver como el segundo juego de parámetros (Listado 4.27) logra mejorar notablemente el resultado del primero (Listado 4.26). El hecho de ir construyendo la serie incrementalmente facilita, como podemos ver, que la población se habitúe al primer período de la secuencia y luego lo extrapole al resto, en vez de construir desde el principio una solución que trate de abarcar la totalidad de la secuencia.

En sucesiones con regularidades más o menos claras, como es el caso, vemos que podemos obtener una mejora notable haciendo uso de este mecanismo.

Listado 4.26: Resultado para A189398_meertens_fullseries

```
Final best individual:
Formula: TK(3.83002e+06)%TK(1.34218)^/(-1)K(46.5185)/K(
1.29725e+06)P[3]+/+TK(2.78396e+06)%TK(11746.5)*/(-1)K(1
882)*K(1.30679e+06)P[3]+/%+#
(((((P[3])+(K(1306788.12255)))/((K(1882.00143266))*((-1
))))%(((K(11746.5275713))*(T))/((K(2783960.97452))%(T))
))+((((P[3])+(K(1297247.17751)))/((K(46.5184947489))/((
-1))))+(((K(1.34218338027))^(T))/((K(3830022.90558))%(T
)))))#
          ^                    ^                    ^
    ^                    ^     ^                    ^
  ^          ^                    ^                    ^
      ^                    ^     ^                    ^
     ^

12345555555555555555555555554445555555555555555555555
55433345555555555555555555554445555555555555555555554
32223455555555555555555555555544455555555555555555555
55554333455555555555555555555544455555555555555555555
543210
Score: 46046 evaluated over a length of 50 out of 50

Ran in 1704.42 seconds
for target data: A189398_meertens_fullseries
```

Listado 4.27: Resultado para A189398_meertens_stepup

```
Final best individual:
Formula: TK(213214)%OK(11114.9)+*K(594771)1*TK(1)+-%TK(
213929)%K(0.999357)P[2]/^OK(1)^+%#
((((K(1))^(0))+(((P[2])/(K(0.999356739131)))^((K(213929
```

```
.233978))%(T))))%((((K(1))+(T))-((1)*(K(594771.439668))
))%(((K(11114.8980315))+(0))*((K(213214.41639))%(T)))))
#
          ^    ^         ^                      ^
          ^       ^          ^    ^    ^
  ^                     ^    ^                  ^
^
1234444444444333455555555555555555555555555444555555555
5555555555554322234555555555544455555555555555555555555
4333455555555555555555555554445555555555555555555554321
0
Score: 26316 evaluated over a length of 50 out of 50

Ran in 1615.78 seconds
for target data: A189398_meertens_stepup
```

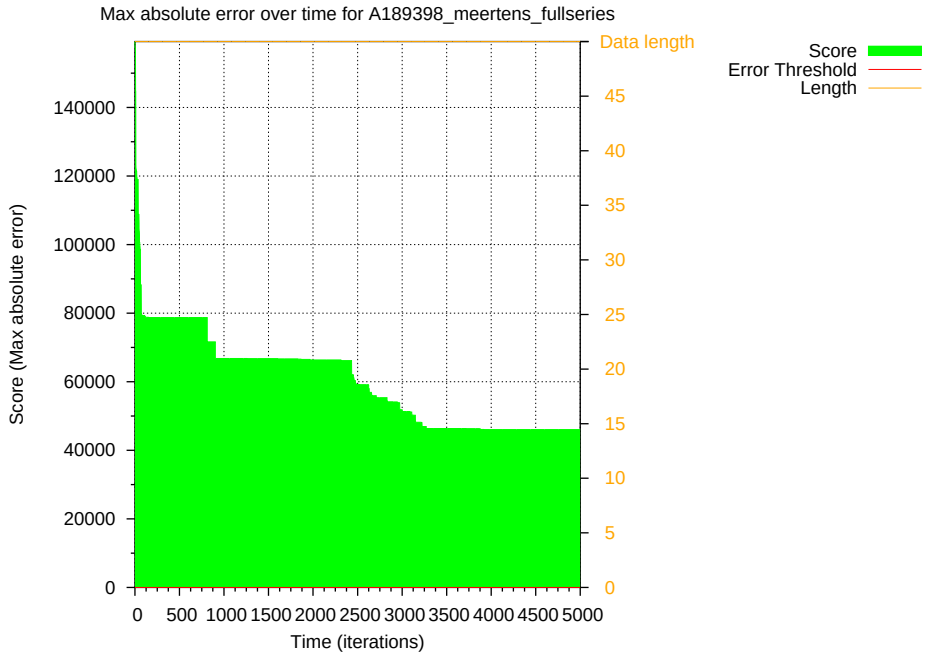

Figura 4.25: Evolución de A189398_meertens_fullseries

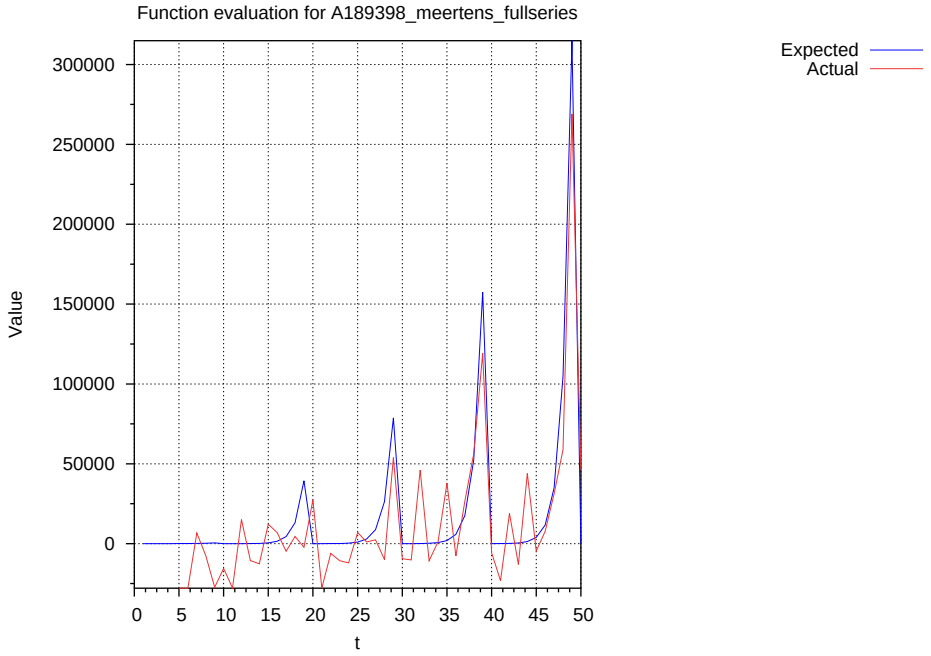

Figura 4.26: Resultado de A189398_meertens_fullseries

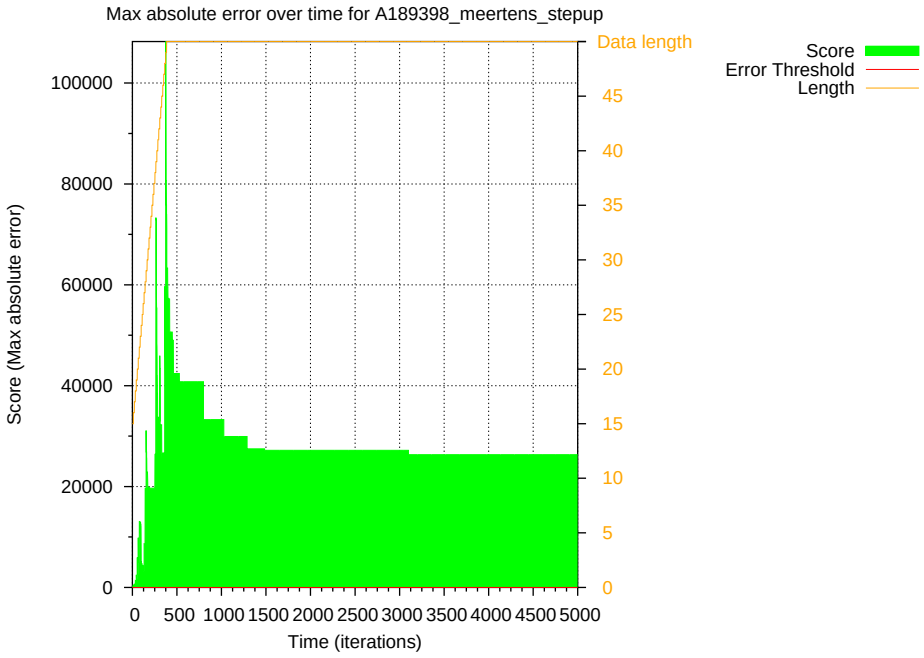

Figura 4.27: Evolución de A189398_meertens_stepup

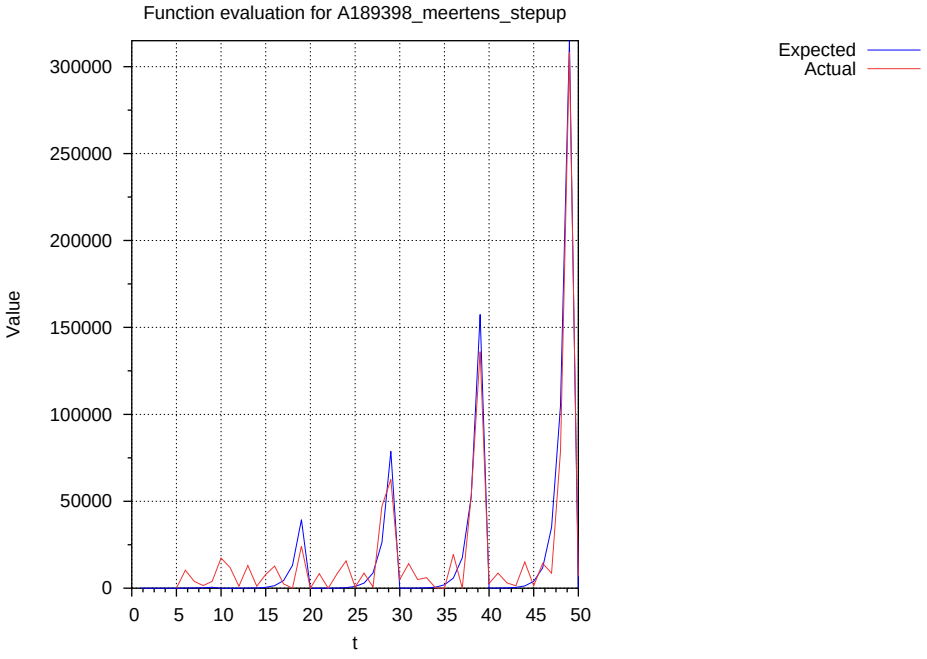

Figura 4.28: Resultado de A189398_meertens_stepup

Apéndice A

Clases auxiliares

Existen una serie de módulos auxiliares empleados en el programa.

A.1. AuxFuncs

Este módulo contiene una serie de funciones auxiliares empleadas en el código principal:

Listado A.1: auxFuncs.h

```
1  ////////////////////////////////////////////////////////////////////////
2  /// File "auxFuncs.h"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Author: Daniel Dominguez Catena
5  ////////////////////////////////////////////////////////////////////////
6
7  #ifndef AUXFUNCS_H
8  #define AUXFUNCS_H
9
10 #include <algorithm>
11 #include <iostream>
12 #include <random>
13 #include <sstream>
14 #include <stack>
15 #include <time.h>
16 #include "constants.h"
17 #include "parameters.h"
18 #include "types.h"
19
20 #define RANDOMEXCLUSIVE(low, high) (rand() % (high - 1 - low) + low
       + 1)
21 #define RANDOM(low, high) ((high < low) ? high : rand() % (high -
       low + 1) + low)
22
23 using namespace std;
24
25 namespace GRGA
```

```
26 {
27
28   inline void printFormulaPostFix(fType * f)
29   {
30     cout << "Formula: ";
31     unsigned int * cur = f->expression;
32     eSymbol s;
33
34     while (*cur != kEnd)
35       {
36         s = static_cast<eSymbol>(*cur);
37         switch (s)
38           {
39           case kInd:
40             cout << mStrings.at(s);
41             cout << "(" << f->parameters[cur - f->expression] << ")
   ";
42             break;
43           case kPrev:
44             cout << mStrings.at(s);
45             cout << "[" << f->prev[cur - f->expression] << "]";
46             break;
47           default:
48             cout << mStrings.at(s);
49             break;
50           }
51         cur++;
52       }
53     cout << "#" << endl;
54   }
55
56   inline void printFormula(fType * f)
57   {
58
59     printFormulaPostFix(f);
60
61     std::ostringstream levels;
62     std::ostringstream operators;
63     std::ostringstream output;
64     int level = 1;
65     std::stack<eSymbol> ops;
66     std::stack<unsigned int> rest;
67     rest.push(1);
68     output << "(";
69     while(levels.tellp() < output.tellp())
70       {
71         levels << level;
72         operators << ' ';
73       }
74
75     unsigned int *cur = &f->expression[f->length - 2];
76     eSymbol s;
77
78     while (cur >= f->expression)
79       {
80         s = static_cast<eSymbol>(*cur);
81         if (s > kOperatorsBin && s < kOperatorsBinEnd)
82           {
83             ops.push(s);
84             rest.push(2);
85             output << "(";
86             level++;
```

```
87            levels << level;
88            operators << ' ';
89          }
90        else
91          {
92            switch (s)
93              {
94              case kInd:
95                output << mStrings.at(s);
96                output.precision(12);
97                output << "(" << f->parameters[cur - f->expression]
       << ")";
98                break;
99              case kPrev:
100               output << mStrings.at(s);
101               output << "[" << f->prev[cur - f->expression] <<
      "]";
102               break;
103             default:
104               output << mStrings.at(s);
105               break;
106             }
107           while(levels.tellp() < output.tellp())
108             {
109               levels << level;
110               operators << ' ';
111             }
112           rest.top()--;
113
114           while((rest.size() > 0) && (rest.top() == 0))
115             {
116               ops.pop();
117               rest.pop();
118               output << ")";
119               levels << level;
120               operators << ' ';
121               level--;
122               if (rest.size() > 0)
123                 rest.top()--;
124             }
125           if (!rest.empty() && rest.top() == 1)
126             {
127               s = ops.top();
128               output << ")" << mStrings.at(s) << "(";
129               levels << level;
130               operators << ' ';
131               while(levels.tellp() < ((int) output.tellp() - 1))
132                 {
133                   levels << level;
134                   operators << '^';
135                 }
136               levels << level;
137               operators << ' ';
138             }
139         }
140       cur--;
141     }
142
143   output << "#";
144   levels << 0;
145   operators << '^';
146   cout << output.str() << endl;
```

```
147     cout << operators.str() << endl;
148     cout << levels.str() << endl;
149   }
150
151   inline bool isOperator(unsigned int s)
152   {
153     return (s > kOperatorsBin) && (s < kOperatorsBinEnd);
154   }
155
156   inline int getRandomNode(fType * f)
157   {
158     int point = 1;
159     int cutNode = RANDOM(1, (f->length)/2 - 1);
160     while (cutNode > 0)
161       {
162         while (!isOperator(f->expression[point]))
163           {
164             point++;
165           }
166         point++;
167         cutNode--;
168       }
169     point--;
170     return point;
171   }
172
173   // Random utilities
174   inline void initRand()
175   {
176     time_t seconds;
177     time(&seconds);
178     srand((unsigned int) seconds);
179   }
180
181   static std::random_device rd;
182   static std::mt19937 gen(rd());
183   static std::normal_distribution<> dist01(0.0, 1.0);
184
185   // Cleaners
186   inline void clearFType(fType * f)
187   {
188     free(f->expression);
189     free(f->prev);
190     free(f->parameters);
191     delete f;
192   }
193
194   inline void clearIType(iType * i)
195   {
196     clearFType(i->formula);
197     delete i;
198   }
199
200   inline void clearPType(pType * p)
201   {
202     for(auto it = p->begin(); it != p->end(); it++)
203       {
204         clearIType(* it);
205       }
206     delete p;
207   }
208
```

```
209   inline void clearIESType(iESType * i)
210   {
211     free(i->control);
212     free(i->parameters);
213     delete i;
214   }
215
216   inline bool compareIESType(iESType * first, iESType * second)
217   {
218     return (first->score < second->score);
219   }
220
221   inline bool compareIType(iType * first, iType * second)
222   {
223     return (first->score > second->score);
224   }
225
226   inline void sortPType(pType * p)
227   {
228     std::sort(p->begin(), p->end(), compareIType);
229   }
230
231 }
232
233
234
235 #endif // AUXFUNCS_H
```

A.2. Plot

En esta librería incluimos el código que se gestiona la salida de gráficos hacia gnuplot.

Listado A.2: plot.h

```
1  ////////////////////////////////////////////////////////////////////
2  /// File "plot.h"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Author: Daniel Dominguez Catena
5  ////////////////////////////////////////////////////////////////////
6
7  #ifndef GRGA_PLOT_H
8  #define GRGA_PLOT_H
9
10 #include "types.h"
11
12 namespace GRGA
13 {
14   namespace Plot
15   {
16     void initializePlots();
17     void closePlots();
18     void plotIteration(int i, scorereal *scores, scorereal *
       thresholdBar, scorereal *lengths, iType *bestind);
19     void plotFinalResults(int i, scorereal *scores, scorereal *
       thresholdBar, scorereal *lengths, iType *bestind);
```

```
20      void plotEvaluation(xreal *results);
21    }
22
23  }
24
25  #endif // GRGA_PLOT_H
```

Listado A.3: plot.cpp

```
1   ////////////////////////////////////////////////////////////////////////
2   /// File "tournament.cpp"
3   /// Project GRGA: Generalized Regression based on Genetic
        Algorithms
4   /// Author: Daniel Dominguez Catena
5   ////////////////////////////////////////////////////////////////////////
6
7   #include <iostream>
8   #include <sstream>
9   #include "direction.h"
10  #include "evaluation.h"
11  #include "parameters.h"
12  #include "plot.h"
13
14  using namespace GRGA;
15
16  void Plot::initializePlots()
17  {
18    std::string errorMode = "Mean";
19    if (p["errorMode"].i == kErrorMax)
20      errorMode = "Max";
21
22    std::string errorFunction = "absolute";
23    if (p["errorFunction"].i == kEFuncRel)
24      errorFunction = "relative";
25
26    std::ostringstream scoreOSS;
27    scoreOSS << "Score (" << errorMode << " " << errorFunction << "
        error)";
28
29    Direction::h2DPlot =
30        gpc_init_2d (const_cast<char*>("Evolution"),             //
        Plot title
31                     const_cast<char*>("Time (iterations)"),
                 // X-Axis label
32                     const_cast<char*>(scoreOSS.str().c_str()),
               // Y-Axis label
33                     GPC_AUTO_SCALE,          // Scaling mode
34                     GPC_SIGNED,              // Sign mode
35                     GPC_MULTIPLOT,           // Multiplot / fast
        plot mode
36                     GPC_KEY_ENABLE);         // Legend / key mode
37    if (Direction::h2DPlot == NULL)                     // Plot
        creation failed - e.g is server running ?
38      {
39        printf ("\nPlot creation failure. Please ensure gnuplot is
        located on your system path\n");
40        exit (-1);
41      }
42    Evaluation::h2DPlot =                               // Initialize
        plot
```

```
43        gpc_init_2d (const_cast<char*>("Results"),             //
      Plot title
44                     const_cast<char*>("t"),                  // X-
      Axis label
45                     const_cast<char*>("value"),             // Y-
      Axis label
46                     GPC_AUTO_SCALE,          // Scaling mode
47                     GPC_SIGNED,              // Sign mode
48                     GPC_MULTIPLOT,           // Multiplot / fast
      plot mode
49                     GPC_KEY_ENABLE);         // Legend / key mode
50    if (Evaluation::h2DPlot == NULL)                    // Plot
      creation failed - e.g is server running ?
51      {
52        printf ("\nPlot creation failure. Please ensure gnuplot is
      located on your system path\n");
53        exit (-1);
54      }
55  }
56
57  void Plot::closePlots()
58  {
59    if (Direction::h2DPlot != NULL)
60      gpc_close(Direction::h2DPlot);
61    if (Evaluation::h2DPlot != NULL)
62      gpc_close(Evaluation::h2DPlot);
63
64    system ("rm *.gpdt");
65  }
66
67  void Plot::plotIteration(int i, scorereal *scores, scorereal *
      thresholdBar, scorereal *lengths, iType *bestind)
68  {
69    std::string errorMode = "Mean";
70    if (p["errorMode"].i == kErrorMax)
71      errorMode = "Max";
72
73    std::string errorFunction = "absolute";
74    if (p["errorFunction"].i == kEFuncRel)
75      errorFunction = "relative";
76
77    fprintf (Direction::h2DPlot->pipe, "clear\n");
78    fprintf (Direction::h2DPlot->pipe, "set title \"%s %s error over
      time for %s\"\n",
79             errorMode.c_str(), errorFunction.c_str(),
80             targetData->name.c_str()); // Set the plot title
81    fprintf (Direction::h2DPlot->pipe, "set yrange [0:*]\n");
82    fprintf (Direction::h2DPlot->pipe, "set y2range [0:%d]\n",
      targetData->length);
83    fprintf (Direction::h2DPlot->pipe, "set y2tics 0, 5, %d textcolor
       rgb \"orange\"\n",
84             targetData->length);
85    fprintf (Direction::h2DPlot->pipe, "set y2tics add (\"Data length
      \" %d)\n", targetData->length);
86    gpc_plot_2ds (Direction::h2DPlot,              // Plot handle
87                  scores,                 // Dataset
88                  i + 1,         // Number of data points
89                  const_cast<char*>("Score"),            // Dataset
      title
90                  0,                 // Minimum X value
91                  i,     // Maximum X value
92                  //"line pt 7 ps 0.5",  // Plot type
```

```
93                const_cast<char*>("filledcurve y1=0"),
94                const_cast<char*>("green"),         // Colour
95                GPC_NEW);               // Add plot
96    gpc_plot_2ds (Direction::h2DPlot,             // Plot handle
97                thresholdBar,                 // Dataset
98                2,         // Number of data points
99                const_cast<char*>("Error Threshold"),           //
       Dataset title
100               0,                 // Minimum X value
101               i,     // Maximum X value
102               const_cast<char*>("line pt 7 ps 0.5"),  // Plot
      type
103               const_cast<char*>("red"),         // Colour
104               GPC_ADD);               // Add plot
105   gpc_plot_2ds (Direction::h2DPlot,             // Plot handle
106               lengths,                // Dataset
107               i + 1,         // Number of data points
108               const_cast<char*>("Length"),            // Dataset
      title
109               0,                 // Minimum X value
110               i,     // Maximum X value
111               const_cast<char*>("line pt 7 ps 0.5"),  // Plot
      type
112               const_cast<char*>("orange \" axes x1y2 \""),
        // Colour
113               GPC_ADD);               // Add plot
114
115   Evaluation::score(bestind->formula, true);
116 }
117
118 void Plot::plotFinalResults(int i, scorereal * scores, scorereal *
      thresholdBar, scorereal * lengths, iType * bestind)
119 {
120   if (i >= p["numIters"].i)
121     i = p["numIters"].i - 1;
122
123   std::string errorMode = "Mean";
124   if (p["errorMode"].i == kErrorMax)
125     errorMode = "Max";
126
127   std::string errorFunction = "absolute";
128   if (p["errorFunction"].i == kEFuncRel)
129     errorFunction = "relative";
130
131   std::ostringstream scoreOSS;
132   scoreOSS << "Score (" << errorMode << " " << errorFunction << "
      error)";
133
134   Direction::h2DPlot =
135       gpc_init_2d (const_cast<char*>("Evolution"),            //
      Plot title
136                    const_cast<char*>("Time (iterations)"),
             // X-Axis label
137                    const_cast<char*>(scoreOSS.str().c_str()),
           // Y-Axis label
138                    GPC_AUTO_SCALE,          // Scaling mode
139                    GPC_SIGNED,              // Sign mode
140                    GPC_MULTIPLOT,           // Multiplot / fast
      plot mode
141                    GPC_KEY_ENABLE);         // Legend / key mode
142   if (Direction::h2DPlot == NULL)                    // Plot
      creation failed - e.g is server running ?
```

```
143     {
144       fprintf (stderr, "\nPlot creation failure. Please ensure
        gnuplot is located on your system path\n");
145       exit (1);
146     }
147
148   fprintf (Direction::h2DPlot->pipe, "\n");
149   fprintf (Direction::h2DPlot->pipe, "set terminal postscript color
         solid\n"); // Output
150   fprintf (Direction::h2DPlot->pipe, "set output \"./results/%
        s_evolution.ps\"\n",
151           targetData->name.c_str());
152
153   fprintf (Direction::h2DPlot->pipe, "set title \"%s %s error over
        time for %s\"\n",
154           errorMode.c_str(), errorFunction.c_str(),
155           targetData->name.c_str()); // Set the plot title
156   fprintf (Direction::h2DPlot->pipe, "set yrange [0:*]\n");
157   fprintf (Direction::h2DPlot->pipe, "set y2range [0:%d]\n",
        targetData->length);
158   fprintf (Direction::h2DPlot->pipe, "set y2tics 0, 5, %d textcolor
         rgb \"orange\"\n",
159           targetData->length);
160   fprintf (Direction::h2DPlot->pipe, "set y2tics add (\"Data length
        \" %d) ", targetData->length);
161   gpc_plot_2ds (Direction::h2DPlot,              // Plot handle
162                 scores,                 // Dataset
163                 i + 1,          // Number of data points
164                 const_cast<char*>("Score"),            // Dataset
        title
165                 0,                  // Minimum X value
166                 i,     // Maximum X value
167                 //"line pt 7 ps 0.5",  // Plot type
168                 const_cast<char*>("filledcurve y1=0"),
169                 const_cast<char*>("green"),         // Colour
170                 GPC_NEW,              // Add plot
171                 false);               // Do not plot yet
172   gpc_plot_2ds (Direction::h2DPlot,              // Plot handle
173                 thresholdBar,                // Dataset
174                 2,          // Number of data points
175                 const_cast<char*>("Error Threshold"),            //
         Dataset title
176                 0,                  // Minimum X value
177                 i,     // Maximum X value
178                 const_cast<char*>("line pt 7 ps 0.5"),  // Plot
        type
179                 const_cast<char*>("red"),          // Colour
180                 GPC_ADD,              // Add plot
181                 true);                // Do not plot yet
182   gpc_plot_2ds (Direction::h2DPlot,              // Plot handle
183                 lengths,                 // Dataset
184                 i + 1,          // Number of data points
185                 const_cast<char*>("Length"),            // Dataset
        title
186                 0,                  // Minimum X value
187                 i,     // Maximum X value
188                 const_cast<char*>("line pt 7 ps 0.5"),  // Plot
        type
189                 const_cast<char*>("orange\" axes x1y2 \""),
         // Colour
190                 GPC_ADD,               // Add plot
191                 true);               // Now you can plot
```

```
192
193   if (Direction::h2DPlot != NULL)
194     gpc_close(Direction::h2DPlot);
195
196   Evaluation::h2DPlot =                                // Initialize
      plot
197     gpc_init_2d (const_cast<char*>("Results"),             //
      Plot title
198                  const_cast<char*>("t"),                  // X-
      Axis label
199                  const_cast<char*>("Value"),             // Y-
      Axis label
200                  GPC_AUTO_SCALE,          // Scaling mode
201                  GPC_SIGNED,              // Sign mode
202                  GPC_MULTIPLOT,           // Multiplot / fast
      plot mode
203                  GPC_KEY_ENABLE);         // Legend / key mode
204   if (Evaluation::h2DPlot == NULL)                     // Plot
      creation failed - e.g is server running ?
205     {
206       printf ("\nPlot creation failure. Please ensure gnuplot is
      located on your system path\n");
207       exit (1);
208     }
209
210   fprintf (Evaluation::h2DPlot->pipe, "\n");
211   fprintf (Evaluation::h2DPlot->pipe, "set terminal postscript
      color solid\n"); // Set the plot
212   fprintf (Evaluation::h2DPlot->pipe, "set output \"./results/%
      s_result.ps\"\n", targetData->name.c_str());
213   fprintf (Evaluation::h2DPlot->pipe, "set title \"Function
      evaluation for %s\"\n",
214            targetData->name.c_str()); // Set the plot title
215   Evaluation::score(bestind->formula, true);
216
217   if (Evaluation::h2DPlot != NULL)
218     gpc_close(Evaluation::h2DPlot);
219
220   system ("rm *.gpdt");
221 }
222
223 void Plot::plotEvaluation(xreal * results)
224 {
225   if (GRGA_PLOT_RESULTS == 1 )
226     {
227       if (Evaluation::h2DPlot == NULL)
228         {
229           Evaluation::h2DPlot =                                //
      Initialize plot
230             gpc_init_2d (const_cast<char*>("Results"),
      // Plot title
231                          const_cast<char*>("t"),
      // X-Axis label
232                          const_cast<char*>("Value"),             //
       Y-Axis label
233                          GPC_AUTO_SCALE,          // Scaling mode
234                          GPC_SIGNED,              // Sign mode
235                          GPC_MULTIPLOT,           // Multiplot /
      fast plot mode
236                          GPC_KEY_ENABLE);         // Legend / key
      mode
```

```
237           if (Evaluation::h2DPlot == NULL)                    //
      Plot creation failed - e.g is server running ?
238           {
239             printf ("\nPlot creation failure. Please ensure
      gnuplot is located on your system path\n");
240             exit (1);
241           }
242         }
243
244       fprintf (Evaluation::h2DPlot->pipe, "clear\n");
245     }
246
247   gpc_plot_2dx (Evaluation::h2DPlot,              // Plot handle
248                 targetData->data,                // Dataset
249                 p["currentTargetLength"].i,        // Number of
      data points
250                 const_cast<char*>("Expected"),            // Dataset
       title
251                 1,                  // Minimum X value
252                 p["currentTargetLength"].i,     // Maximum X value
253                 const_cast<char*>("line pt 7 ps 0.5"),  // Plot type
254                 const_cast<char*>("blue"),         // Colour
255                 GPC_NEW, false);              // Add plot
256
257   gpc_plot_2dx (Evaluation::h2DPlot,              // Plot handle
258                 results,                 // Dataset
259                 p["currentTargetLength"].i - p["maxPrev"].i,
       // Number of data points
260                 const_cast<char*>("Actual"),            // Dataset
      title
261                 p["maxPrev"].i + 1,                 // Minimum X
      value
262                 p["currentTargetLength"].i,     // Maximum X value
263                 const_cast<char*>("line pt 7 ps 0.5"),  // Plot type
264                 const_cast<char*>("light-red"),         // Colour
265                 GPC_ADD);              // Add plot
266
267 }
```

A.3. Gnuplot_c

Incluímos como referencia la librería de interfaz con Gnuplot que hemos empleado. En concreto, hemos realizado algunas modificaciones para garantizar la ejecución de plots repetidos sin generar "basura" y hemos duplicado la función principal de impresión para que acepte tanto dobles como flotantes.

Además, hemos reducido el código eliminando las funciones no utilizadas para reducir el espacio empleado en esta memoria.

La librería original puede encontrarse en http://www.numerix-dsp.com/files/. Está liberada para cualquier uso, sin garantías por parte del autor.

Listado A.4: gnuplot_c.h

```
1 ////////////////////////////////////////////////////////////////////
2 /// File "gnuplot_c.h"
```

```
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Original Developer: Numerix-DSP
5  /// Webpage: http://www.numerix-dsp.com/files/
6  /// Modified by Daniel Dominguez Catena
7  ////////////////////////////////////////////////////////////////////

8
9  #ifndef GNUPLOT_H
10 #define GNUPLOT_H
11
12 // Gnuplot/C interface library header file
13 // Please ensure that the system path includes an entry for the
       gnuplot binary folder
14
15 #ifndef GPC_DEBUG
16   #define GPC_DEBUG                 0           // Set to '1' to
       enable Gnuplot text debug output, '0' otherwise
17 #endif
18
19 #include <stdio.h>
20 #include <stdlib.h>
21 #include <sys/stat.h>
22
23 #define MAX_NUM_GRAPHS          50              // Maximum number
       of graphs (multiplots) in a plot
24
25 #define GPC_AUTO_SCALE          0.0             // Auto scaling
26 #define GPC_IMG_AUTO_SCALE      0               // Auto scaling for
        images
27
28 #define CANVAS_WIDTH            1100
29 #define CANVAS_HEIGHT           800
30 #define PLOT_LMARGIN            (170.0/CANVAS_WIDTH)   // 512 pixel
        X axis
31 #define PLOT_RMARGIN            (682.0/CANVAS_WIDTH)
32
33 #define GPC_NO_ERROR            0               // No error
34 #define GPC_ERROR               1               // Error
35
36 #define GPC_FALSE               0               // False flag
37 #define GPC_TRUE                1               // True flag
38
39 #define GPC_END_PLOT            NULL            // Flag to indicate
        end of the plot
40
41 enum gpcMultiFastMode                           // Multiplot / fast
        plot modes
42 {
43   GPC_MULTIPLOT = 0,
44   GPC_FASTPLOT
45 };
46
47 enum gpcKeyMode                                 // Legend / Key
       mode
48 {
49   GPC_KEY_DISABLE = 0,
50   GPC_KEY_ENABLE
51 };
52
53 enum gpcPlotSignMode                            // Sign modes -
       signed, positive, negative
```

```
54 {
55   GPC_SIGNED = 0,
56   GPC_POSITIVE,
57   GPC_NEGATIVE
58 };
59
60 enum gpcNewAddGraphMode                          // New / Add graph
       modes
61 {
62   GPC_NEW = 0,
63   GPC_ADD,
64   GPC_FINISH
65 };
66
67 enum gpcPoleZeroMode                             // Pole zero plot
       modes
68 {
69   GPC_COMPLEX_POLE = 0,
70   GPC_CONJUGATE_POLE,
71   GPC_COMPLEX_ZERO,
72   GPC_CONJUGATE_ZERO
73 };
74
75                                                   // Spectrogram and
       image colour palettes
76 #define GPC_MONOCHROME "set palette defined (0 'black', 1 'white')"
77 #define GPC_COLOUR "set palette defined (0 'black', 1 'blue', 2 '
       red', 3 'yellow', 4 'white')"
78
79
80 typedef struct                                   // Complex data
       type
81 {
82   double  real;
83   double  imag;
84 } ComplexRect_s;
85
86
87 typedef struct
88 {
89   char filename [40];                            // Graph filename
90   char title [80];                               // Graph title
91   char formatString [40];                        // Graph format
       string "lines", "points" etc
92 } h_GPC_Graph;
93
94
95 typedef struct
96 {
97   FILE *pipe;                                    // Pipe to Gnuplot
       instance
98   int numberOfGraphs;                            // Number of graphs
       on this plot
99   h_GPC_Graph graphArray [MAX_NUM_GRAPHS];       // Array of graphs
100  char plotTitle[80];                            // Plot title
101  int filenameRootId;                            // Multiplot
       filename root Id
102  enum gpcMultiFastMode multiFastMode;           // Multiplot / fast
       plot mode
103  double scalingMode;                            // Scaling mode /
       dimension for XY and PZ graphs
104  long xAxisLength;                              // X axis length
```

```
    for spectrogram plots and images
105   long yAxisLength;                          // Y axis length
        for spectrogram plots and images
106   double xMin;                               // Minimum value of
        X axis - used for axis labels
107   double xMax;                               // Maximum value of
        X axis - used for axis labels
108   double yMin;                               // Minimum value of
        Y axis - used for axis labels
109   double yMax;                               // Maximum value of
        Y axis - used for axis labels
110   long numberOf2ndAxisPlotted;               // Number of
        columns for spectrogram plot
111   int multiPlot2DFlag;                       // Flag set when 2D
        graph created
112 } h_GPC_Plot;
113
114
115 h_GPC_Plot *gpc_init_2d (char *plotTitle,    // Plot title
116   char *xLabel,                              // X axis label
117   char *yLabel,                              // Y axis label
118   double scalingMode,                        // Scaling mode
119   enum gpcPlotSignMode signMode,             // Sign mode -
        signed, positive, negative
120   enum gpcMultiFastMode multiFastMode,       // Multiplot / fast
        plot mode
121   enum gpcKeyMode keyMode);                  // Legend / key
        mode
122
123 int gpc_plot_2df(h_GPC_Plot *plotHandle,     // Plot handle
124   float *pData,                              // Dataset pointer
125   int graphLength,                           // Dataset length
126   char *pDataName,                           // Dataset title
127   double xMin,                               // Minimum X value
128   double xMax,                               // Maximum X value
129   char *plotType,                            // Plot type - "
        lines", "points", "impulses", "linespoints"
130   char *pColour,                             // Colour - Use
        gnuplot> show colornames to see available colours
131   enum gpcNewAddGraphMode addMode,           // Add / new mode
132   bool plot = true);                         // Whether or not
        to plot
133
134
135 int gpc_plot_2dd(h_GPC_Plot *plotHandle,     // Plot handle
136   double *pData,                             // Dataset pointer
137   int graphLength,                           // Dataset length
138   char *pDataName,                           // Dataset title
139   double xMin,                               // Minimum X value
140   double xMax,                               // Maximum X value
141   char *plotType,                            // Plot type - "
        lines", "points", "impulses", "linespoints"
142   char *pColour,                             // Colour - Use
        gnuplot> show colornames to see available colours
143   enum gpcNewAddGraphMode addMode,           // Add / new mode
144   bool plot = true);                         // Whether or not
        to plot
145
146 void gpc_close (h_GPC_Plot *);               // Plot handle
147
148 #endif
```

Listado A.5: gnuplot_c.cpp

```
1  ////////////////////////////////////////////////////////////////////
2  /// File "gnuplot_c.h"
3  /// Project GRGA: Generalized Regression based on Genetic
       Algorithms
4  /// Original Developer: Numerix-DSP
5  /// Webpage: http://www.numerix-dsp.com/files/
6  /// Modified by Daniel Dominguez Catena
7  ////////////////////////////////////////////////////////////////////
8
9  // Gnuplot/C interface library
10 // Please ensure that the system path includes an entry for the
       gnuplot binary folder
11
12 #include <math.h>
13 #include <stdio.h>
14 #include <stdlib.h>
15 #include <string.h>
16 #include "gnuplot_c.h"
17
18 #define GPC_REPLOT 0                // This is being used for
       testing multiplot/replot which has some strange side effects so
        is not used in the release
19
20 #if defined (_MSC_VER)                          // Defined by
       Microsoft compilers
21   #include <windows.h>
22   #if (GPC_DEBUG == 1)
23     #define GNUPLOT_CMD "pgnuplot -persist"                    //
       Window pipe version
24     // #define GNUPLOT_CMD "gnuplot -persist"
       // Do not pipe the text output to null so that it can be used
       for debugging
25     // #define GNUPLOT_CMD "gnuplot -persist > debug.log 2>&1"
       // Pipe the text output to debug.log for debugging
26   #else
27     #define GNUPLOT_CMD "gnuplot -persist > /nul 2>&1"          //
       Pipe the text output to null for higher performance
28   #endif
29   #define popen _popen
30   #define pclose _pclose
31   #define sleep Sleep
32   // #pragma warning(disable:4996)                                //
       -D "_CRT_SECURE_NO_WARNINGS=1"
33 #else
34   #if (GPC_DEBUG == 1)
35     #define GNUPLOT_CMD "gnuplot"      // Do not pipe the text
       output to null so that it can be used for debugging
36   #else
37     //#define GNUPLOT_CMD "gnuplot -persist > debug.log"
       // Pipe the text output to debug.log for debugging
38     //#define GNUPLOT_CMD "gnuplot -persist > /dev/null 2>&1"
       // Pipe the text output to null for higher performance
39     #define GNUPLOT_CMD "gnuplot > /dev/null 2>&1"     // Pipe the
       text output to null for higher performance
40   #endif
41 #endif
42
43 h_GPC_Plot *gpc_init_2d (char *plotTitle,
44   char *xLabel,
```

```
45    char *yLabel,
46    double scalingMode,
47    enum gpcPlotSignMode signMode,
48    enum gpcMultiFastMode multiFastMode,
49    enum gpcKeyMode keyMode)
50
51  {
52          h_GPC_Plot *plotHandle;
            // Create plot
53
54          plotHandle = (h_GPC_Plot*) malloc (sizeof (h_GPC_Plot));
            // Malloc plot and check for error
55          if (plotHandle == NULL)
56          {
57                  return (plotHandle);
58          }
59
60    plotHandle->pipe = popen (GNUPLOT_CMD, "w");                    //
        Open pipe to Gnuplot and check for error
61    if (plotHandle->pipe == NULL)
62    {
63      free (plotHandle);
64                  return (plotHandle);
65    }
66
67    strcpy (plotHandle->plotTitle, plotTitle);                      //
        Set plot title in handle
68    plotHandle->multiFastMode = multiFastMode;                      //
        Set multiplot / fastplot mode in handle
69
70    if (multiFastMode == GPC_MULTIPLOT)
71    {
72      plotHandle->multiPlot2DFlag = GPC_TRUE;                       //
        This is a 2D plot - used in gpc_close ()
73    }
74    else
75    {
76      plotHandle->multiPlot2DFlag = GPC_FALSE;                      //
        This is a 2D plot - used in gpc_close ()
77    }
78
79    fprintf (plotHandle->pipe, "set term wxt 0 title \"%s\" size %d,
        %d\n", plotHandle->plotTitle, CANVAS_WIDTH, CANVAS_HEIGHT); //
        Set the plot
80    fprintf (plotHandle->pipe, "set lmargin at screen %4.8lf\n",
        PLOT_LMARGIN); // Define the margins so that the graph is 512
        pixels wide
81    fprintf (plotHandle->pipe, "set rmargin at screen %4.8lf\n",
        PLOT_RMARGIN);
82    fprintf (plotHandle->pipe, "set border back\n");                //
        Set border behind plot
83
84    fprintf (plotHandle->pipe, "set xlabel \"%s\"\n", xLabel);      //
        Set the X label
85    fprintf (plotHandle->pipe, "set ylabel \"%s\"\n", yLabel);      //
        Set the Y label
86    fprintf (plotHandle->pipe, "set grid x y\n");                   //
        Turn on the grid
87    fprintf (plotHandle->pipe, "set tics out nomirror\n");          //
        Tics format
88    fprintf (plotHandle->pipe, "set mxtics 4\n");
89    fprintf (plotHandle->pipe, "set mytics 2\n");
```

```
90
91    if (keyMode == GPC_KEY_ENABLE)
92    {
93      fprintf (plotHandle->pipe, "set key out vert nobox\n");     //
        Legend / key location
94    }
95    else
96    {
97      fprintf (plotHandle->pipe, "unset key\n");                  //
        Disable legend / key
98    }
99
100   if (scalingMode == GPC_AUTO_SCALE)                            //
        Set the Y axis scaling
101   {
102     fprintf (plotHandle->pipe, "set autoscale  yfix\n");        //
        Auto-scale Y axis
103   }
104   else
105   {
106     if (signMode == GPC_SIGNED)                                 //
        Signed numbers (positive and negative)
107     {
108       fprintf (plotHandle->pipe, "set yrange [%lf:%lf]\n", -
      scalingMode, scalingMode);
109     }
110     else if (signMode == GPC_POSITIVE)                          // 0
       to +ve Max
111     {
112       fprintf (plotHandle->pipe, "set yrange [0.0:%lf]\n",
      scalingMode);
113     }
114     else                                                        //
      GPC_NEGAIVE - -ve Min to 0
115     {
116       fprintf (plotHandle->pipe, "set yrange [%lf:0.0]\n", -
      scalingMode);
117     }
118   }
119
120   fflush (plotHandle->pipe);                                    //
      flush the pipe
121
122   return (plotHandle);
123 }
124
125 int gpc_plot_2df(h_GPC_Plot *plotHandle,
126   float *pData,
127   int graphLength,
128   char *pDataName,
129   double xMin,
130   double xMax,
131   char *plotType,
132   char *pColour,
133   enum gpcNewAddGraphMode addMode,
134   bool plot)
135
136 {
137   int   i;
138   FILE  *gpdtFile;
139   char  tmpFilename [30];
140   struct stat   fileStatBuffer;
```

```
141
142   if (plotHandle->multiFastMode == GPC_MULTIPLOT)        //
        GPC_MULTIPLOT
143   {
144     if (addMode == GPC_NEW)             // GPC_NEW
145     {
146
147       // fprintf (plotHandle->pipe, "set autoscale x\n");
        // Auto-scale Y axis
148       plotHandle->numberOfGraphs = 0;
149
150       i = -1;
151       do                                                    //
        Create a unique local filename - Note this is NOT MT safe !
152       {
153         i++;
154         sprintf (tmpFilename, "%d-0.gpdt", i);
155       } while (stat (tmpFilename, &fileStatBuffer) == 0);
156       plotHandle->filenameRootId = i;
157     }
158     else                                // GPC_ADD
159     {
160       plotHandle->numberOfGraphs++;
161       if (plotHandle->numberOfGraphs >= (MAX_NUM_GRAPHS - 1))   //
        Check we haven't overflowed the maximum number of graphs
162       {
163         return (GPC_ERROR);
164       }
165     }
166
167     sprintf (plotHandle->graphArray[plotHandle->numberOfGraphs].
        filename, "%d-%d.gpdt", plotHandle->filenameRootId, plotHandle
        ->numberOfGraphs);
168     sprintf (plotHandle->graphArray[plotHandle->numberOfGraphs].
        title, "%s", pDataName);
169     sprintf (plotHandle->graphArray[plotHandle->numberOfGraphs].
        formatString, "%s lc rgb \"%s\"", plotType, pColour);
170
171     gpdtFile = fopen (plotHandle->graphArray[plotHandle->
        numberOfGraphs].filename, "w");    // Open temporary files
172     for (i = 0; i < graphLength; i++)                           //
        Write data to intermediate file
173     {
174       fprintf (gpdtFile, "%lf %lf\n", xMin + ((((double)i) * (xMax
        - xMin)) / ((double)(graphLength - 1))), pData[i]);
175     }
176     fclose (gpdtFile);
177
178     if (plot)
179     {
180       fprintf (plotHandle->pipe, "plot \"%s\" using 1:2 title \"%s
        \" with %s", plotHandle->graphArray[0].filename, plotHandle->
        graphArray[0].title, plotHandle->graphArray[0].formatString);
        // Send start of plot and first plot command
181       for (i = 1; i <= plotHandle->numberOfGraphs; i++)
        // Send individual plot commands
182       {
183   #if defined (_MSC_VER)                                  // Defined
        by Microsoft compilers
184         fprintf (plotHandle->pipe, ", \\\r \"%s\" using 1:2 title
        \"%s\" with %s", plotHandle->graphArray[i].filename, plotHandle
        ->graphArray[i].title, plotHandle->graphArray[i].formatString);
```

```
      // Set plot format
185   #else
186         fprintf (plotHandle->pipe, ", \\\n \"%s\" using 1:2 title
        \"%s\" with %s", plotHandle->graphArray[i].filename, plotHandle
        ->graphArray[i].title, plotHandle->graphArray[i].formatString);
        // Set plot format
187   #endif
188       }
189       fprintf (plotHandle->pipe, "\n");
      // Send end of plot command
190     }
191   }
192   else                                                    //
        GPC_FASTPLOT
193   {
194     if (addMode == GPC_NEW)
195     {
196       fprintf (plotHandle->pipe, "set xrange [%lf:%lf]\n", xMin -
      (0.5 * ((xMax - xMin) / (graphLength - 1))),
197                                                             xMax +
      (0.5 * ((xMax - xMin) / (graphLength - 1))));  // Set length of
       X axis
198     }
199
200     fprintf (plotHandle->pipe, "plot '-' using 1:2 title \"%s\"
      with %s lc rgb \"%s\"\n", pDataName, plotType, pColour);  //
      Set plot format
201     for (i = 0; i < graphLength; i++)                          //
      Copy the data to gnuplot
202     {
203       fprintf (plotHandle->pipe, "%lf %lf\n", xMin + ((((double)i)
      * (xMax - xMin)) / ((double)(graphLength - 1))),
204                                               pData[i]);
205     }
206     fprintf (plotHandle->pipe, "e\n");                         //
      End of dataset
207   }                                                  // End of
        GPC_MULTIPLOT/GPC_FASTPLOT
208
209   fflush (plotHandle->pipe);                                 //
        Flush the pipe
210
211   #if GPC_DEBUG
212     sleep (1);                                              //
        Slow down output so that pipe doesn't overflow when logging
        results
213   #endif
214
215   return (GPC_NO_ERROR);
216 }
217
218 int gpc_plot_2dd(h_GPC_Plot *plotHandle,
219   double *pData,
220   int graphLength,
221   char *pDataName,
222   double xMin,
223   double xMax,
224   char *plotType,
225   char *pColour,
226   enum gpcNewAddGraphMode addMode,
227   bool plot)
228
```

```
229 {
230   int   i;
231   FILE  *gpdtFile;
232   char  tmpFilename [30];
233   struct stat   fileStatBuffer;
234 
235   if (plotHandle->multiFastMode == GPC_MULTIPLOT)        //
        GPC_MULTIPLOT
236   {
237     if (addMode == GPC_NEW)             // GPC_NEW
238     {
239 
240       // fprintf (plotHandle->pipe, "set autoscale x\n");
      // Auto-scale Y axis
241       plotHandle->numberOfGraphs = 0;
242 
243       i = -1;
244       do                                                          //
      Create a unique local filename - Note this is NOT MT safe !
245       {
246         i++;
247         sprintf (tmpFilename, "%d-0.gpdt", i);
248       } while (stat (tmpFilename, &fileStatBuffer) == 0);
249       plotHandle->filenameRootId = i;
250     }
251     else                                // GPC_ADD
252     {
253       plotHandle->numberOfGraphs++;
254       if (plotHandle->numberOfGraphs >= (MAX_NUM_GRAPHS - 1))   //
      Check we haven't overflowed the maximum number of graphs
255       {
256         return (GPC_ERROR);
257       }
258     }
259 
260     sprintf (plotHandle->graphArray[plotHandle->numberOfGraphs].
      filename, "%d-%d.gpdt", plotHandle->filenameRootId, plotHandle
      ->numberOfGraphs);
261     sprintf (plotHandle->graphArray[plotHandle->numberOfGraphs].
      title, "%s", pDataName);
262     sprintf (plotHandle->graphArray[plotHandle->numberOfGraphs].
      formatString, "%s lc rgb \"%s\"", plotType, pColour);
263 
264     gpdtFile = fopen (plotHandle->graphArray[plotHandle->
      numberOfGraphs].filename, "w");    // Open temporary files
265     for (i = 0; i < graphLength; i++)                           //
      Write data to intermediate file
266     {
267       fprintf (gpdtFile, "%lf %lf\n", xMin + ((((double)i) * (xMax
      - xMin)) / ((double)(graphLength - 1))), pData[i]);
268     }
269     fclose (gpdtFile);
270 
271     if (plot)
272     {
273       fprintf (plotHandle->pipe, "plot \"%s\" using 1:2 title \"%s
      \" with %s", plotHandle->graphArray[0].filename, plotHandle->
      graphArray[0].title, plotHandle->graphArray[0].formatString);
      // Send start of plot and first plot command
274       for (i = 1; i <= plotHandle->numberOfGraphs; i++)
      // Send individual plot commands
275       {
```

```
276   #if defined (_MSC_VER)                                    // Defined
        by Microsoft compilers
277           fprintf (plotHandle->pipe, ", \\\r \"%s\" using 1:2 title
        \"%s\" with %s", plotHandle->graphArray[i].filename, plotHandle
        ->graphArray[i].title, plotHandle->graphArray[i].formatString);
          // Set plot format
278   #else
279           fprintf (plotHandle->pipe, ", \\\n \"%s\" using 1:2 title
        \"%s\" with %s", plotHandle->graphArray[i].filename, plotHandle
        ->graphArray[i].title, plotHandle->graphArray[i].formatString);
          // Set plot format
280   #endif
281       }
282       fprintf (plotHandle->pipe, "\n");
        // Send end of plot command
283     }
284   }
285   else                                                      //
        GPC_FASTPLOT
286   {
287     if (addMode == GPC_NEW)
288     {
289       fprintf (plotHandle->pipe, "set xrange [%lf:%lf]\n", xMin -
        (0.5 * ((xMax - xMin) / (graphLength - 1))),
290                                                               xMax +
        (0.5 * ((xMax - xMin) / (graphLength - 1))));  // Set length of
         X axis
291     }
292
293     fprintf (plotHandle->pipe, "plot '-' using 1:2 title \"%s\"
        with %s lc rgb \"%s\"\n", pDataName, plotType, pColour);  //
        Set plot format
294     for (i = 0; i < graphLength; i++)                          //
        Copy the data to gnuplot
295     {
296       fprintf (plotHandle->pipe, "%lf %lf\n", xMin + ((((double)i)
        * (xMax - xMin)) / ((double)(graphLength - 1))),
297                                               pData[i]);
298     }
299     fprintf (plotHandle->pipe, "e\n");                         //
        End of dataset
300   }                                                  // End of
        GPC_MULTIPLOT/GPC_FASTPLOT
301
302   fflush (plotHandle->pipe);                                 //
        Flush the pipe
303
304 #if GPC_DEBUG
305   sleep (1);                                                //
        Slow down output so that pipe doesn't overflow when logging
        results
306 #endif
307
308   return (GPC_NO_ERROR);
309 }
310
311 void gpc_close (h_GPC_Plot *plotHandle)
312 {
313   int i;
314
315   fprintf (plotHandle->pipe, "exit\n");                      //
        Close GNUPlot
```

```
316   pclose (plotHandle->pipe);                                    //
        Close the pipe to Gnuplot
317   if (plotHandle->multiPlot2DFlag == GPC_TRUE)                  //
        If this is a 2D plot we need to delete the temporary files
318   {
319     for (i = 0; i <= plotHandle->numberOfGraphs; i++)           //
        Remove all temporary files
320     {
321       remove (plotHandle->graphArray[i].filename);
322     }
323   }
324
325   free (plotHandle);                                            //
        Free the plot
326   plotHandle = NULL;
327
328 }
```

Apéndice B

Compilación

Los requisitos del programa son:

- Compilador GCC[1] con soporte para C++ en el estándar C++11 y OpenMP (testeado para la versión 4.8.2).
- Librería Gnuplot[2] (testeado para la versión 4.6).

Se puede ejecutar la compilación con el pequeño script del Listado B.1.

Los ficheros necesarios se encuentran en los listados Listado 3.2 a Listado 3.20 y Listado A.1 a Listado A.5. Todos los ficheros se deben llamar con el nombre indicado y colocarse en la misma carpeta que el script de compilación. En esa misma carpeta se creará una carpeta `bin` donde se colocarán los resultados de la compilación, y dentro de ella dos carpetas `input` y `results`, donde se especificarán las tareas a realizar (véase la Sección 3.4 y se recogerán los resultados.

Listado B.1: compile.sh

```
1  CXX_FLAGS="-fopenmp -D_REENTRANT -std=c++11 -lgomp -fopenmp"
2
3  g++ -c parameters.cpp $CXX_FLAGS
4  g++ -c stack.cpp $CXX_FLAGS
5  g++ -c gp.cpp $CXX_FLAGS
6  g++ -c evaluation.cpp $CXX_FLAGS
7  g++ -c generation.cpp $CXX_FLAGS
8  g++ -c plot.cpp $CXX_FLAGS
9  g++ -c direction.cpp $CXX_FLAGS
10 g++ -c tournament.cpp $CXX_FLAGS
11 g++ -c gnuplot_c.cpp $CXX_FLAGS
12 g++ -c es.cpp $CXX_FLAGS
13 g++ -c main.cpp $CXX_FLAGS
14
15 g++ parameters.o stack.o gp.o evaluation.o generation.o plot.o
       direction.o tournament.o es.o gnuplot_c.o main.o -fopenmp -
       D_REENTRANT -std=c++11 -lgomp -fopenmp -o grga
16
```

[1] http://gcc.gnu.org/
[2] http://www.gnuplot.info/

```
17 mv *.o ./bin/
18 mv grga ./bin/
```

Bibliografía

[1] Richard S. Bird, *Meertens number*. Journal of Functional Programming, vol. 8, no. 1, pp. 83-88. 1998.

[2] Diday, E., *The symbolic approach in clustering and related methods of Data Analysis*. H.H. Bock (ed.), Classification and Related Methods of Data Analysis. Amsterdam: North-Holland, 1987.

[3] Diday, E., *Introduction à l'approche symbolique en analyse des données. Première Journées Symbolique-Numérique*. Université Paris IX Dauphine, Diciembre 1987.

[4] R. Dubcakova, *Eureqa: software review*, Genetic Programming and Evolvable Machines, vol. 12, no. 2, pp. 173–178, Junio 2011.

[5] Eureqa: http://www.nutonian.com/

[6] David A. Freedman, *Statistical Models: Theory and Practice*. Cambridge University Press, 2005.

[7] J. Fan, I. Gijbels, *1.1 From linear regression to nonlinear regression*. Local Polynomial Modelling and Its Applications. Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, 1996.

[8] Gödel, Kurt, *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I*. Monatsheft für Math und Physik, no. 38, pp. 173-198. 1931.

[9] H. Iba, T. Sato, and H. de Garis. *Recombination guidance for numerical genetic programming*. 1995 IEEE Conference on Evolutionary Computation, vol. 1, pp. 97–102, Perth, Australia, 29 Noviembre - 1 Diciembre 1995. IEEE Press.

[10] A. G. Ivakhnenko, *Heuristic Self-Organization in Problems of Engineering Cybernetics*. Automatica, vol. 6, pp. 207-219. Pergamon Press, 1970.

http://www.gmdh.net/articles/history/heuristic.pdf

[11] J. Koza, *Human-competitive results produced by genetic programming*. Genetic Programming and Evolvable Machines, vol. 11, pp. 251-284, 2010.

[12] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[13] J. C. Lagarias, *The 3x + 1 Problem and Its Generalizations*. 1994.

[14] David M. Lane, *Introduction to Linear Regression*. En Online Statistics Education: An Interactive Multimedia Course of Study, edición online disponible en:

http://onlinestatbook.com/2/regression/intro.html

[15] B. McKay, M. J. Willis y G. W. Barton, *Using a tree structured genetic algorithm to perform symbolic regression*. First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA, vol. 414, pp. 487-492, Sheffield, UK, 12-14 Septiembre 1995.

[16] H.P.Ritzema *Frequency and Regression Analysis*. Drainage Principles and Applications, no. 16, pp. 175-224, 1994.

http://www.waterlog.info/pdf/regtxt.pdf

[17] M. Schoenauer, M. Sebag, F. Jouve, B. Lamy, y H. Maitournam, *Evolutionary identification of macro-mechanical models*. Advances in Genetic Programming 2, cap. 23, pp. 467-488. MIT Press, Cambridge, MA, USA, 1996.

[18] S.N. Sivanandam, S. N. Deepa, *1.4 Advantages of Evolutionary Computation*. Introduction to Genetic Algorithms, pp. 9-13, Springer, 2007.

[19] K. C. Sharman, A. I. Esparcia Alcazar, y Y. Li, *Evolving signal processing algorithms by genetic programming*. First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA, vol. 414, pp. 473-480, Sheffield, UK, 12-14 Septiembre 1995.

[20] Sodas2: https://www.info.fundp.ac.be/asso/sodaslink.htm

[21] R. Poli, W. B. Langdon, and N. F. McPhee, *A field guide to genetic programming*. Publicado vía http://lulu.com y libremente disponible en http://www.gp-field-guide.org.uk, 2008. (Con contribuciones de J. R. Koza).

[22] Vanderbilt University, *Robot biologist solves complex problem from scratch*. Octubre 2011.

http://cacm.acm.org/careers/136345-robot-biologist-solves-complex-\
problem-from-scratch/fulltext

[23] Vladislavleva, E.J., *Order of Nonlinearity as a Complexity Measure for Models Generated by Symbolic Regression via Pareto Genetic Programming*. En IEEE Transactions on Evolutionary Computation vol. 13, no. 2, pp. 333-349, Septiembre 2008.